The Unit Graphs Mathematical Framework

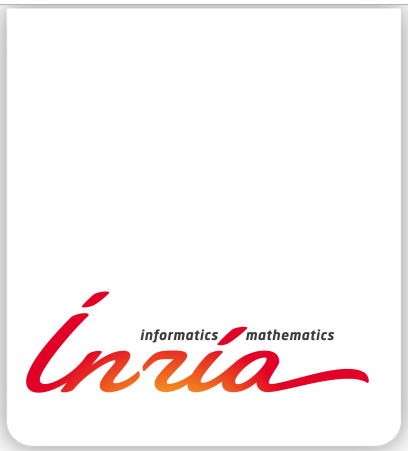
Maxime Lefrançois, Fabien Gandon

The Unit Graphs Mathematical Framework

Maxime Lefrançois, Fabien Gandon

RESEARCH REPORT
N° 8212
February 2013
Project-Team Wimmics

ISSN 0249-6399 ISRN INRIA/RR--8212--FR+ENG



The Unit Graphs Mathematical Framework

Maxime Lefrançois, Fabien Gandon

Project-Team Wimmics

Research Report n° 8212 — version 3 — initial version February 2013 — revised version June 2013 — 67 pages

Abstract:
We are interested in the choice of a graph-based knowledge representation formalism that would allow for the representation, manipulation, query, and reasoning over linguistic knowledge of the Explanatory Combinatorial Dictionary (ECD) of the Meaning-Text Theory (MTT). We show that neither the semantic web formalisms nor the Conceptual Graphs Formalism are suitable for this task, and we justify the introduction of the new Unit Graphs (UGs) framework. We introduce the fundamental concepts of the UGs mathematical framework: the Unit Types hierarchy and the Unit Graphs. We then show how they may be used to represent lexicographic definitions in the ECD. Finally we provide the UGs with reasoning capabilities.

Key-words: Linguistic Knowledge Representation, Meaning-Text Theory, Explanatory and Combinatorial Dictionary, Unit Graphs

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Le formalisme mathématique des graphes d'unités

Résumé : Nous nous intéressons au choix d'un formalisme de représentation des connaissances à base de graphes qui permette de représenter, manipuler, interroger et raisonner sur des connaissances linguistiques du Dictionnaire Explicatif et Combinatoire (DEC) de la Théorie Sens-Texte. Nous montrons que ni les formalismes du web sémantique ni le formalisme des Graphes conceptuels (GC) n'est adapté pour cela, et justifions l'introduction d'un nouveau formalisme dit des Graphes d'Unités. Nous introduisons donc les concepts fondamentaux du formalisme des Graphes d'Unités (GU): la hiérarchie des Types d'Unités et les Graphes d'Unités. Nous montrons ensuite comment on peut les utiliser pour représenter les définitions lexicographiques dans le DEC. Nous permettons finalement le raisonnement dans les GU.

Mots-clés : Représentation de Connaissances Linguistiques, Théorie Sens-Texte, Dictionnaire Explicatif et Combinatoire, Graphes d'Unités

Contents

Glossary

ASlot Actant Slot

ASymbol Actant Symbol

CG Conceptual Graph

CSymbol Circumstantial Symbol

CUT Conjunctive Unit Type

DSemUT Deep Semantic Unit Type

ECD Explanatory Combinatorial Dictionary

KR Knowledge Representation

LexUT Lexical Unit Type

MTT Meaning-Text Theory

OblASlot Obligatory Actant Slot

OptASlot Optional Actant Slot

ProASlot Prohibited Actant Slot

PUT Primitive Unit Type

SemASlot Semantic Actant Slot

SSemUT Surface Semantic Unit Type

SWF Semantic Web Formalism

UG Unit Graph

Notation system

We will borrow and slightly extend the Meaning-Text notation system in this document:

S–	Surface–	D–	Deep–
–Sem–	–Semantic–	–Syn– .	–Syntactic–
–Morph–	–Morphologic–	–Phon–	–Phonologic–
L–	Lexical–	G–	Grammatical–
–U	–Unit	–UT ...	–Unit Type
FALL	a specific LUT	*present*	a specific GUT
$^{(}$ball$^{)}$	a specific SSemUT	$/$ball$^{\backslash}$.	a specific DSemUT
$\ulcorner \text{FALL}_{\text{present}} : * \urcorner$	a specific SemU typed with a conjunct of a LUT and a GUT		

Part I
Introduction

In this research report we are interested in the choice of a graph-based Knowledge Representation (KR) formalism that would allow for the representation, manipulation, query, and reasoning over linguistic knowledge of the the Explanatory Combinatorial Dictionary (ECD), which is the lexicon at the core of the Meaning-Text Theory (MTT) (c.f. for instance Mel'čuk et al., 1999; Mel'čuk, 2006).

We envision two application scenarios of such a formalization:

- In a ECD lexicographic edition oriented project, we could enable the semi-automation of some of the lexicographers tasks. For instance we could check that a set of constraints is satisfied, or we could suggest preliminary drafts of article (e.g., lexical function key-value pairs, lexicographic definition sketches, government pattern).

- We could propose a syntax, which is a formal language based on knowledge engineering standards. Like WordNet today, the linguistic knowledge written with that syntax could be published to the web of linked data[1]. This would support their use as a highly structured lexical resource by consumers of the linked data cloud.

Most past or current projects that consisted in implementing the ECD did so in a lexicographic perspective. For instance projects NADIA-DEC (Sérasset, 1997), Dicouèbe (Polguère, 2000), DicoInfo and DicoEnviro (L'Homme, 2008), DiCE (Alonso Ramos, 2003) for Spanish.

One important example is the RELIEF project (Lux-Pogodalla and Polguère, 2011) which aims at representing a lexical system graph named RLF (Polguère, 2009) where lexical units are interlinked by paradigmatic and syntagmatic links of lexical functions (e.g., Mel'čuk, 1996). In the RELIEF project, the description of Lexical Functions is based on a formalization proposed by Kahane and Polguère (2001). The RELIEF is also based on different formalization works to represent lexicographic definitions, namely: a hierarchy of semantic labels (Polguère, 2011), the markup type that has been developed in the Definiens project (Barque and Polguère, 2008; Barque et al., 2010) to specify genus and specific differences, and the disambiguation of meaningful words in the definition.

One exception is the proprietary linguistic processor ETAP-3 that implements a variety of ECD for Natural Language Processing (Apresian et al., 2003; Boguslavsky et al., 2004). Linguistic knowledge elements are asserted, and linguistic and grammatical rules are directly formalized in first order logic.

Adding to these formalization works, our goal is to propose a formalization from a knowledge engineering perspective, compatible with standard KR formalisms. The term *formalization* here means not only *make non-ambiguous*, but also *make operational*, i.e., *such that it is adapted to logical operations* (e.g., knowledge manipulation, query, reasoning). We thus adopt a knowledge engineering approach applied to the domain of the MTT, and our research question here is *What knowledge representation formalism would be adapted to represent knowledge of the ECD ?*

At first sight, two existing KR formalisms seem to fit this task:

- the semantic web formalisms, because the linked data is built on them;

- the Conceptual Graphs (CGs) formalism (Sowa, 1984; Chein and Mugnier, 2008), as we are to lead logic reasoning on graphs.

[1]The web of data is a W3C initiative, highly active today, `http://linkeddata.org`

Our research question may thus be decomposed in three sub-questions that we address in this research report:

- Are these formalisms adapted to represent knowledge of the ECD ?
- If *no*, then how can we define one that is adapted to the domain of the MTT ?
- What are the direct implications for the MTT ?

The rest of this research report is organized as follows. In this part, we will first overview how a lexicographer of the RELIEF project may represent lexicographic definitions (§1), and we will justify that neither the semantic web formalisms nor the CGs are adapted to represent the knowledge of the ECD (§2). We will hence state the following choice:

We modify the CGs formalism basis, and define transformations to the RDF syntax for sharing knowledge and publishing over the web of data.

As we will represent linguistic units of different nature (e.g., semantic units, lexical units, grammatical units, words), term *unit* has been chosen to be used in a generic manner, and the result of this adaptation is thus *the Unit Graph (UG) mathematical framework*.

Part II focuses on the core of the UGs framework which is a hierarchy of unit types that have an actantial structure. Part III will be devoted to the UGs *per se*. Part 12 will introduce rules and lexicographic definitions, and detail a possible improvement of the scenario for the RELIEF project.

1 Lexicographic Definitions in the RELIEF project: Current Scenario

The lexicographic edition software developed in the RELIEF project is named MVSDicet. Let us sketch a scenario where Alain, the leader of the project, assigns the task of defining the French lexical unit PEIGNE$_{\mathbf{2A}}$, which is defined in (Mel'čuk et al., 1999) by:

PEIGNE$_{\mathbf{2A}}$: $^{(}$comb$^{)}\equiv^{(}$Weaving tool that a person X uses to untangle object Y$^{)}$.

1. Sophie first seeks for a semantic label in the hierarchy of semantic labels (Polguère, 2011). She chooses $^{/}$outil$^{\backslash}$ ($^{(}$tool$^{)}$).
2. Sophie determines that PEIGNE$_{\mathbf{2A}}$ has two obligatory semantic actants: a person X, and an object Y. She then seeks for a fitting propositional form in a hierarchy that only Alain develops. She may choose: $\sim$ de X [pour Y] ($\sim$ of X [for Y]).
3. Sophie then writes the lexicographic definition marked up with genus and specific differences as in the Definiens project (Barque and Polguère, 2008; Barque et al., 2010). Finally for each of the meaningful words of the lexicographic definition, Sophie specifies the lexical unit of the RLF it refers to.

```
<CC label="outil">outil de tissage</CC>
<PC role="utilisation">que X utilise pour peigner#2 Y</PC>
```

2 Motivations to Introduce a New Knowledge Representation Formalism

At first sight, two existing KR formalisms seem interesting for the MTT. Semantic web formalisms (RDF/S, OWL, SPARQL), because the linked data is built on them, and CGs formalism (Sowa, 1984; Chein and Mugnier, 2008), as we are to lead logical reasoning on graphs. Both formalisms are based on directed labelled graph structures, and some research has been done towards using them to represent dependency structures and knowledge of the lexicon (OWL in (Lefrançois and Gandon, 2011a; Boguslavsky, 2011), CGs at the conceptual level in (Bohnet and Wanner, 2010)).

In this section we answer the following question:

What makes the semantic web formalisms and the CGs formalisms not directly adapted to the representation of the knowledge of the ECD ?

Let us first recall that for a specific Lexical Unit L, Mel'čuk (2004, p.5) distinguishes by considering L in language (i.e., in the lexicon), or in speech (i.e., in an utterance). The KR formalisms also do this distinction using types. Objects of the represented domain are named instances (or objects, or individuals), and are typed (or classified).

2.1 Semantic Web Formalisms

There is a world wide deployment of the semantic web formalisms, and the RDF[2] syntax is the standard for structured data exchange over the web of linked data. The expressivity of RDF would be sufficient to represent the knowledge of the ECD. Yet, the semantics of RDF, in the logical sense, is limited to that of oriented labelled multi-graphs, and we wish also to enable the manipulation and reasoning over linguistic knowledge of the ECD. We thus need to introduce more semantics with RDFS[3] or OWL[4], while keeping the expressivity as low as possible to keep good computational properties. OWL introduces semantics with axioms[5] and classes and relation constructors[6]. Yet RDFS and OWL only support binary relations, which is not the case of most lexemes having arguments (most verbs, adjectives and nouns, such as METHOD or COMB). One could use reification of n-ary relations[7], but then no semantics is attributed to such relations.

The ULiS project (Lefrançois and Gandon, 2011a) did envision an architecture for a multilingual knowledge base compatible with the MTT and based on OWL. In the ULiS project, axioms and class constructors are used in order to make each lexical unit support the projection of its lexicographic definition over itself. We identified three major problems with the usage of OWL for that.

- For each lexical unit definition, one need to introduce as many new semantic relations as there exists nodes in the definition graph of the lexical unit. This implies an overload of useless relations.
- These relations must be combined using the sub relation chains axioms `SubObjectPropertyOf( ObjectPropertyChain( OPE`$_1$ `...OPE`$_n$ `) OPE)`, in order to little by little project the lexical unit definition graph on the lexical unit itself. The OWL2 DL roughly corresponds to the $\mathcal{SROIQ}$ description logics fragment. In this fragment, the hierarchy of roles (=

[2]RDF - Resource Description Framework, `http://w3.org/RDF/`
[3]RDFS - RDF Schema, `http://www.w3.org/TR/rdf-schema/`
[4]OWL - Web Ontology Language, `http://www.w3.org/TR/owl2-overview/`
[5]e.g., Sub-class `SubClassOf(CE`$_1$ `CE`$_2$`)`; Functional Relation: `FunctionalObjectProperty(OPE)`
[6]e.g., Exact cardinality `ObjectExactCardinality(n OPE)`
[7]N-ary relations on the Semantic Web, `http://www.w3.org/TR/swbp-n-aryRelations`

primary relations) must be *regular*[8] (c.f., Rudolph, 2011, ,§2.1) in order to guarantee the decidability of basic reasoning problems. We will show that this regularity is not ensured in the small example ontology given by Lefrançois and Gandon (2011a). We defined the lexical unit DIE, and wanted to define the lexical unit KILL roughly as follows: *Causing ones death*. We then wanted to model that the time of an event *kill* is the same as the time of the event *die* of the person killed. To do so, we defined a new role `hasKillTime`, and used two relation axioms: `SubObjectPropertyOf( ObjectPropertyChain(hasEvent hasTime ) hasKillTime )`, and `SubObjectPropertyOf( hasKillTime hasTime )`. The first axiom implies that `hasKillTime` is a non-simple role, and this combined with the second axiom implies that `hasTime` is also a non-simple role. Yet one needs to have a strict partial order $\prec$ over the set of non-simple roles. First axiom implies that `hasTime` $\prec$ `hasKillTime`, and second axiom implies that `hasKillTime` $\prec$ `hasTime`. Thus as $\prec$ is required to be strict, the role hierarchy is not regular and this example ontology slips into OWL Full and undecidability. The intuitive meaning of this is that `hasKillTime` is defined using `hasTime` in the first axiom, and vice versa in the second axiom. This restriction is thus too important to represent definitions of the ECD.

- Finally, the semantics of the Sub Property Chains axiom makes that inference is anyways possible only in one direction (sub property and not equivalence). This means that when there is the definition of the lexical unit in the graph one may infer that there is the lexical unit, but not the other way around.

One alternative to represent lexicographic definitions of lexical units would be to use two reciprocal CONSTRUCT SPARQL rules. We then face the problem of rule languages and their compatibility with OWL (c.f., Krisnadhi et al., 2011), that led to no consensus nor standard today.

These different problems led us to consider another formalism to represent knowledge of the ECD. We nevertheless want to be able to export these knowledge in RDF to exchange them over the web of linked data.

2.2 The Conceptual Graphs (CGs) Formalism

The CGs formalism (Sowa, 1984; Mugnier and Chein, 1996; Chein and Mugnier, 2008) has many similarities with the MTT. In their basic version, CGs represent typed instances interconnected by typed n-ary relations. Actually, the main goal of Sowa was natural language processing, and his inspiration came from the same source as the MTT founders: Tesnière (1959). Two of the most important similarities for our work are the following:

- Sowa (1989) early suggested to introduce type definitions of concepts and relations that do look similar to lexical units definitions in the ECD. Later on Leclère (1998) also worked on the possibility to reason with type and concept definitions.
- The MTT massively uses rules, for instance to declare correspondences between utterances at different representation levels. Rules and their semantics, in the logical sense, have been thoroughly studied in the CGs literature.

One more asset of CGs is the fact that there are transformations between CGs and RDF/S (c.f., Corby et al., 2000; Baget et al., 2010). One could use these transformations to rewrite CGs in RDF for publication over the web of linked data. Moreover, one could adapt the architecture described in the ULiS (Lefrançois and Gandon, 2011b) project to CGs, that envisioned a

[8] c.f. for instance, `http://www.w3.org/TR/owl2-syntax#The_Restrictions_on_the_Axiom_Closure`

MTT-compliant pivot-based multilingual knowledge base architecture, using the semantic web formalisms, hence:

> Through the [Universal Linguistic System (ULiS)], a user could interact with an interlingual knowledge base (IKB) in controlled natural language. Linguistic resources themselves [would be] part of a specific IKB: The Universal Lexical Knowledge base (ULK), so that actors may enhance their controlled natural language, through requests in controlled natural language.

Yet it is also not natural to represent the knowledge of the ECD using the CGs. Here are two reasons for that:

- A semantic-unit may be represented as a concept type as it is instantiated in utterance semantic representations. On the other hand, if the associated lexical unit is predicative and has Semantic Actant Slots (SemASlots), then the semantic unit may dually be represented as a n-ary relation, so that its instances link other sense units. The CGs don't offer a natural representation of this duality. In fact, in CGs, one must alternate concepts and relations, and a semantic representation of an utterance such as the one in figure 1 can't be directly represented by a CG.
- SemASlots of a lexical unit may differ from those of the lexical unit from which its sense derives[9] (c.f., Mel'čuk, 2004). Yet in the CGs, the inheritance mechanism of relation types, that models the fact that *a relation type is more specific than another*, is constrained so that two relations with different arities must be incomparable. One thus cannot use this natural inheritance mechanism to model the specialization of meanings.

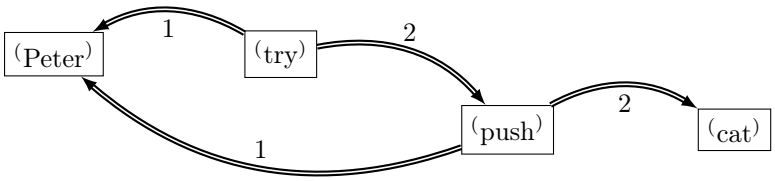


Figure 1: Illustration of the duality concept/relation of semantic units in the MTT, semantic representation of utterance *Peter tries to push the cat*.

[9]For instance, semantic unit (rain) is more specific than (fall) but the meaning of *what falls* and *where it falls from* is fixed to (water drops) and (sky/cloud) (Mel'čuk, 2004).

2.3 The new Unit Graphs Formalism

To sum up, neither the semantic web formalisms nor the CGs formalism allow for a the natural representation of a hierarchy of unit types that may have actant slots, which is the basic knowledge of the ECD. As the CG formalism is the closest to the MTT, we decide to use it as a starting point for designing a new graph-based formalism adapted to the representation of the knowledge of the ECD.

As we are to represent linguistic units of different nature (e.g., semantic units, lexical units, grammatical units, words), we choose to use the term *unit* in a generic manner and name the result of this adaptation *Unit Graphs (UGs) framework*.

In the rest of this research report, we will answer the following research question: *How shall we revisit the CGs formalism so as to make it adapted to represent knowledge of the ECD ?* This question may be decomposed in two sub-questions:

- What mathematical structure for a hierarchy of unit types that may have actant slots (part II)?
- What is a UG, and how to use them to represent advanced concepts of the MTT (part III) ?
- How can we represent rules and lexicographic definitions in the UGs framework ? (part 12) ?

Part II
Unit Types

This part introduces a mathematical structure for a hierarchy of unit types that may have actant slots.

First, for a specific Lexical Unit L, (Mel'čuk, 2004) distinguishes considering L in language (i.e., in the lexicon), or in speech (i.e., in an utterance). KR formalisms and the UGs formalism also do this distinction using types. In this research report and in the UGs formalism, we do establish a distinction between:

- Unit types (e.g., sense unit type, lexical unit type), which will be described in the ECD;
- Units (e.g., sense unit, lexical unit), which will be represented in the UGs.

This distinction corresponds to that of classes vs. instances in RDFS and OWL, and concept type vs. individual in CGs.

Unlike the CGs formalism, every type may be considered both as a concept (i.e., it has instances), and as a n-ary relation (i.e., it links instances). Unit types will specify through actant slots how their instances (i.e., units) shall be linked to other units in a UG. Unit types and their actantial structure are described in a structure called hierarchy and denoted by $\mathcal{T}$.

3 Primitive Unit Types (PUTs)

This section first introduces basic building blocks of the unit types hierarchy which are called *Primitive Unit Types (PUTs)* (§3.1), and then introduces notions used to characterize PUTs as n-ary relation. Whether they are semantic, lexical or grammatical, unit types have Actant Slots (ASlots) (§3.2) with symbols. Moreover, ASlots may be optional, obligatory or prohibited (§3.3), and are signed so that the type of the filler of a ASlot is specified (§3.4).

3.1 Definition of PUTs

$\mathcal{T}$ first contains a finite set of declared *Primitive Unit Types (PUTs)*, denoted by T_D. This set is to contain linguistic PUTs of different nature (e.g., semantic, lexical, grammatical).

Definition 3.1 (Set of declared PUT)**.** A set of *declared Primitive Unit Type (PUT)* is a finite set denoted by T_D.

Second, so as to define actantial structures, $\mathcal{T}$ contains a set of binary relation symbols called *Actant Symbols (ASymbols)*, denoted by $\boldsymbol{S}_{\mathcal{T}}$. $\boldsymbol{S}_{\mathcal{T}}$ contains numbers for the semantic unit types, and other "classical" symbols for the other levels under consideration (e.g, roman numbers I to VI for the MTT's Deep Syntactic level).

Definition 3.2 (Set of ASymbols)**.** An *Actant Symbols (ASymbols) set* is a finite set of binary relation symbols denoted by $\boldsymbol{S}_{\mathcal{T}}$.

Then, no matter whether it is semantic, lexical or grammatical, a PUT has a set (that may be empty) of ASlots whose symbols are chosen in the set of ASymbols. Some ASlots may be obligatory, other optional (Mel'čuk, 2004, p.24), and we postulate that some may be prohibited too. For instance the Lexical Unit Type (LexUT) TO EAT has at least one obligatory semantic ASlot which is for the animal that eats, and an optional semantic ASlot which is for the container the animal eats in.

In order to represent these different types of ASlots, and so that their presence in the hierarchy of Unit Types is coherent, we introduce three bijective mappings over the set of ASymbols: $\boldsymbol{\gamma}$, $\boldsymbol{\gamma_1}$, and $\boldsymbol{\gamma_0}$.

Definition 3.3 (Radices, Obligant, Prohibent of ASymbols)**.** The actantial structure of PUTs is driven by:

- $\boldsymbol{\Gamma}$ is the set of *radices*[10]; $\boldsymbol{\gamma}$ is a bijection from $\boldsymbol{S_\mathcal{T}}$ to $\boldsymbol{\Gamma}$ that assigns to each ASymbol $s \in \boldsymbol{S_\mathcal{T}}$ its *radix*[11] PUT $\boldsymbol{\gamma}(s)$ that introduces an ASlot of symbol s.
- $\boldsymbol{\Gamma_1}$ is the set of *obligant*[12]; $\boldsymbol{\gamma_1}$ is a bijection from $\boldsymbol{S_\mathcal{T}}$ to $\boldsymbol{\Gamma_1}$ that assigns to each ASymbol $s \in \boldsymbol{S_\mathcal{T}}$ its *obligat*[13] PUT $\boldsymbol{\gamma_1}(s)$ that has its ASlot of symbol s obligatory.
- $\boldsymbol{\Gamma_0}$ is the set of *prohibent*[14]; $\boldsymbol{\gamma_0}$ is a bijection from $\boldsymbol{S_\mathcal{T}}$ to $\boldsymbol{\Gamma_0}$ that assigns to each ASymbol $s \in \boldsymbol{S_\mathcal{T}}$ its *prohibet*[15] PUT $\boldsymbol{\gamma_0}(s)$ that has its ASlot of symbol s prohibited.
- $\boldsymbol{\Gamma}$, $\boldsymbol{\Gamma_1}$, and $\boldsymbol{\Gamma_0}$ are pairwise disjoint and disjoint from the set of declared PUTs T_D.

The set of PUTs is denoted by $\boldsymbol{T}$ and is the disjoint union of the set of declared PUTs T_D, the set of radices $\boldsymbol{\Gamma}$, the set of obligant $\boldsymbol{\Gamma_1}$, the set of prohibent $\boldsymbol{\Gamma_0}$, the *prime universal PUT* $\top$ and the *prime absurd PUT* $\bot$.

Definition 3.4 (PUT Set)**.** A *PUT Set* is a disjoint union denoted

$$\boldsymbol{T} \stackrel{\text{def}}{=} T_D \uplus \boldsymbol{\Gamma} \uplus \boldsymbol{\Gamma_1} \uplus \boldsymbol{\Gamma_0} \uplus \{\top\} \uplus \{\bot\}$$

where:

- T_D is a finite set of *declared PUTs*;
- $\boldsymbol{\Gamma}$ is the set of radices;
- $\boldsymbol{\Gamma_1}$ is the set of obligant;
- $\boldsymbol{\Gamma_0}$ is the set of prohibent;
- $\top$ is the *prime universal PUT*;
- $\bot$ is the *prime absurd PUT*.

We introduce an inheritance mechanism for the PUTs which models a specialization relation. This takes the form of a pre-order $\lesssim$ over the set $\boldsymbol{T}$. $t_1 \lesssim t_2$ models the fact that t_1 is more specific than t_2, e.g., $/\text{scissors}\backslash \lesssim /\text{tool}\backslash$ means that $/\text{scissors}\backslash$ is more specific than $/\text{tool}\backslash$. The pre-order over the set of PUTs is induced by a set $\boldsymbol{C_T}$ of comparisons of PUTs.

Definition 3.5 (Pre-order over $\boldsymbol{T}$)**.** The PUT set is pre-ordered by a relation $\lesssim$, which is induced by a set

$$\boldsymbol{C_T} \stackrel{\text{def}}{=} C_A \cup C_\top \cup C_\bot \cup C_{\boldsymbol{\Gamma_1}} \cup C_{\boldsymbol{\Gamma_0}} \subseteq \boldsymbol{T}^2$$

where:

[10] radices is a latin word, the plural of radix, and means ⟨roots⟩.
[11] radix is a latin word that means ⟨root⟩.
[12] obligant is the conjugated form of the latin verb obligo, 3p plur. indic. pres., ⟨they make mandatory⟩.
[13] obligat is the conjugated form of the latin verb obligo, 3p sing. indic. pres., ⟨it makes mandatory⟩.
[14] prohibent is the conjugated form of the latin verb prohibeo, 3p plur. indic. pres., ⟨they prohibit⟩.
[15] prohibet is the conjugated form of the latin verb prohibeo, 3p sing. indic. pres., ⟨it prohibits⟩.

- $C_A \subseteq \boldsymbol{T}^2$ is the set of *asserted comparisons*;
- $C_\top \stackrel{\text{def}}{=} \{(\top, t)\}_{t \in \boldsymbol{T}}$ ensures that $\top$ is more general than every PUT;
- $C_\bot \stackrel{\text{def}}{=} \{(t, \bot)\}_{t \in \boldsymbol{T}}$ ensures that $\bot$ is more specific than every PUT;
- $C_{\boldsymbol{\Gamma_1}} \stackrel{\text{def}}{=} \{(\boldsymbol{\gamma}(s), \boldsymbol{\gamma_1}(s))\}_{s \in \boldsymbol{S_\mathcal{T}}}$ ensures that for every ASymbol the obligat is more specific than the radix;
- $C_{\boldsymbol{\Gamma_0}} \stackrel{\text{def}}{=} \{(\boldsymbol{\gamma}(s), \boldsymbol{\gamma_0}(s))\}_{s \in \boldsymbol{S_\mathcal{T}}}$ ensures that for every ASymbol the prohibet is more specific than the radix.

$(\boldsymbol{T}, \boldsymbol{C_T})$ is a directed graph on $\boldsymbol{T}$. Let $\boldsymbol{C_T^*}$be the reflexo-transitive closure of $\boldsymbol{C_T}$, i.e., $(t, t') \in \boldsymbol{C_T^*}$ iif t' is a descendant of t in $\boldsymbol{C_T}$. The pre-order relation $\lesssim$ is equal to $\boldsymbol{C_T^*}$, i.e., $\forall t, t' \in \boldsymbol{T}, t' \lesssim t$ iif $(t, t') \in \boldsymbol{C_T^*}$.

By construction, the radix of a specific ASymbol is more general than its obligat and its prohibet.

Proposition 3.1. *For all* $s \in \mathbf{S}_\mathcal{T}$, $\boldsymbol{\gamma_1}(s) \lesssim \boldsymbol{\gamma}(s)$ *and* $\boldsymbol{\gamma_0}(s) \lesssim \boldsymbol{\gamma}(s)$.

Proof. Let $s \in \boldsymbol{S_\mathcal{T}}$.
1) $(\boldsymbol{\gamma}(s), \boldsymbol{\gamma_1}(s)) \in C_{\boldsymbol{\Gamma_1}}$, so $\boldsymbol{\gamma_1}(s) \lesssim \boldsymbol{\gamma}(s)$;
2) $(\boldsymbol{\gamma}(s), \boldsymbol{\gamma_0}(s)) \in C_{\boldsymbol{\Gamma_0}}$, so $\boldsymbol{\gamma_0}(s) \lesssim \boldsymbol{\gamma}(s)$; □

By construction, the set of PUTs is bounded by $\top$, a maximal element of $\boldsymbol{T}$, and $\bot$, a minimal element of $\boldsymbol{T}$:

Proposition 3.2. $\top$ *and* $\bot$ *are respectively a maximal and a minimal element of* $\mathbf{T}$.

Proof. Let us prove that 1) $\top$ is a maximal element of $\boldsymbol{T}$, and 2) $\bot$ is a minimal element of $\boldsymbol{T}$.
1) Let $t \in \boldsymbol{T}$. From definition 3.5, $(\top, t) \in C_\top$.
$C_\top \subseteq \boldsymbol{C_T} \subseteq \boldsymbol{C_T^*}$, so $(\top, t) \in \boldsymbol{C_T^*}$. So $t \lesssim \top$ (def. 3.5) and $\top$ is a maximal element of $\boldsymbol{T}$.
2) Conversely, let $t \in \boldsymbol{T}$. From definition 3.5, $(t, \bot) \in C_\bot$.
$C_\bot \subseteq \boldsymbol{C_T} \subseteq \boldsymbol{C_T^*}$, so $(t, \bot) \in \boldsymbol{C_T^*}$, so $\bot \lesssim t$ (def. 3.5) and $\bot$ is a maximal element of $\boldsymbol{T}$. □

Let $\simeq$ be the natural *equivalence relation* over PUTs defined by $t \simeq t' \Leftrightarrow t \lesssim t'$ and $t' \lesssim t$. The set of equivalence classes defines a partition of $\boldsymbol{T}$. Let $t \in \boldsymbol{T}$, we denote $[t]$ the equivalence class to which t belongs, i.e., $[t] \stackrel{\text{def}}{=} \{t' \in \boldsymbol{T} \mid t' \simeq t\}$.

Definition 3.6 (Equivalence PUTs class set)**.** The *equivalence PUTs class set* $\boldsymbol{T}^\sim$ is the quotient set of $\boldsymbol{T}$ by $\simeq$, i.e., $\boldsymbol{T}^\sim \stackrel{\text{def}}{=} \boldsymbol{T}/\simeq = \{[t] \mid t \in \boldsymbol{T}\}$. We denote $t^\sim$ a generic equivalence PUTs class set. We define a partial order $\stackrel{\sim}{\lesssim}$ over $\boldsymbol{T}/\simeq$ with $t_1^\sim \stackrel{\sim}{\lesssim} t_2^\sim$ if and only if $\exists t_1 \in t_1^\sim, t_2 \in t_2^\sim; t_1 \lesssim t_2$.

$\bot$ is the prime absurd PUT and should by definition be the type of no unit. Thus, any PUT that is more specific than $\bot$ should also be absurd. As $\bot$ is also a minimal element of $\boldsymbol{T}$, any PUT which is more specific than $\bot$ is actually equivalent to $\bot$. The equivalence class to which $\bot$ belongs is thus called the set of absurd PUTs and denoted $\bot^\sim$. Any $t \in \bot^\sim$ is said to be absurd and no unit should have this PUT as a type.

Definition 3.7 (Absurd PUT set)**.** The set of absurd PUTs is denoted by $\bot^\sim$ and is the set: $\bot^\sim \stackrel{\text{def}}{=} [\bot]$.

We introduce a graphical representation similar to UML for PUTs. As illustrated in figure 2, a PUT is represented by a box, and the UML specialization relation is reused here.

PUTs may both be considered as a concept (i.e., they have instances), and as a n-ary relation (i.e., they link instances). Next sections are devoted to PUTs seen as relations.

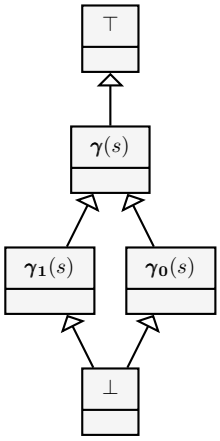


Figure 2: Prototypical hierarchy of PUTs for one ASymbol.

3.2 Actant Slots of a PUT (ASlots)

Then, no matter whether it is semantic, lexical or grammatical, a PUT $t \in \boldsymbol{T}$ has a set (that may be empty) of Actant Slots (ASlots) whose symbols are chosen in the set of Actant Symbols (ASymbols) $\boldsymbol{S_\mathcal{T}}$. Some ASlots may be obligatory, other optional (Mel'čuk, 2004, p.24), and we postulate that some may be prohibited too. For instance the LexUT TO EAT has at least one obligatory semantic ASlot which is for the animal that eats, and an optional semantic ASlot which is for the container the animal eats in. If one specializes the meaning of TO EAT to define a new LexUT, we identify three basic cases that may happen:

- An optional ASlot may become obligatory.
- An optional ASlot may become prohibited, e.g., the container for TO GRAZE;
- A new ASlot may be introduced;

As every ASlot has a symbol, the set of ASlots of a PUT $t \in \boldsymbol{T}$ is defined as the set of their symbols $\boldsymbol{\alpha}(t) \subseteq \boldsymbol{S_\mathcal{T}}$. Formally, the actantial structure of t is defined by the set of radices, obligant and prohibent:

- t has a Actant Slot (ASlot) for every ASymbol whose radix is more general or equivalent to t, and only for those ASymbols;
- t has an Obligatory Actant Slot (OblASlot) for every ASymbol whose obligat is more general or equivalent to t, and only for those ASymbols;
- t has a Prohibited Actant Slot (ProASlot) for every ASymbol whose prohibet is more general or equivalent to t, and only for those ASymbols;

Definition 3.8 (ASlots of a PUT, and valency of a PUT)**.** The set of *Actant Slots (ASlots)* of PUTs is defined through a mapping $\boldsymbol{\alpha}$ from $\boldsymbol{T}$ to $2^{\boldsymbol{S_\mathcal{T}}}$, that associates every PUT $t \in \boldsymbol{T}$ with the set of ASymbols whose radix is more general than t, i.e.,

$$\forall t \in \boldsymbol{T}, \boldsymbol{\alpha}(t) \stackrel{\text{def}}{=} \{s \in \boldsymbol{S_\mathcal{T}} \mid t \lesssim \boldsymbol{\gamma}(s)\}$$

For every $t \in \boldsymbol{T}$, $s \in \boldsymbol{\alpha}(t)$ means that t has a ASlot with ASymbol s. The number of ASlots of a PUT t is denoted the *valency* of t, i.e., $valency(t) \stackrel{\text{def}}{=} |\boldsymbol{\alpha}(t)|$.

Definition 3.9 (OblASlots of a PUT)**.** The set of *Obligatory Actant Slots (OblASlots)* of PUTs is defined through a mapping $\boldsymbol{\alpha_1}$ from $\boldsymbol{T}$ to $2^{\boldsymbol{S_\mathcal{T}}}$, that associates every PUT $t \in \boldsymbol{T}$ with the set of ASymbols whose obligat is more general than t,

$$\forall t \in \boldsymbol{T}, \boldsymbol{\alpha_1}(t) \stackrel{\text{def}}{=} \{s \in \boldsymbol{S_\mathcal{T}} \mid t \lesssim \boldsymbol{\gamma_1}(s)\}$$

For every $t \in \boldsymbol{T}$, $s \in \boldsymbol{\alpha_1}(t)$ means that t has an OblASlot with ASymbol s.

Definition 3.10 (ProASlots of a PUT)**.** The set of *Prohibited Actant Slots (ProASlots)* of PUTs is defined through a mapping $\boldsymbol{\alpha_0}$ from $\boldsymbol{T}$ to $2^{\boldsymbol{S_\mathcal{T}}}$, that associates every PUT $t \in \boldsymbol{T}$ with the set of ASymbols whose prohibet is more general than t,

$$\forall t \in \boldsymbol{T}, \boldsymbol{\alpha_0}(t) \stackrel{\text{def}}{=} \{s \in \boldsymbol{S_\mathcal{T}} \mid t \lesssim \boldsymbol{\gamma_0}(s)\}$$

For every $t \in \boldsymbol{T}$, $s \in \boldsymbol{\alpha_0}(t)$ means that t has an ProASlot with ASymbol s.

Note. Let s be a ASymbols. As a shortcut, instead of "ASlot having symbol s", we simply write: "ASlot s".

Note. In the following, we use "more general" and "more specific" in their weak sense, i.e., respectively "more general or equivalent" and "more specific or equivalent", unless otherwise stated.

As a direct implication of the definitions of ASlots, the set of PUTs that have a ASlot (resp1. OblASlot, resp2. ProASlot) of a given ASymbol $s \in \boldsymbol{S_\mathcal{T}}$ is the set of PUTs that is more specific than the radix (resp1. obligat, resp2. prohibet) of s.

Proposition 3.3. *Let $\downarrow t$ be the smallest lower set of $\mathbf{T}$ containing t, i.e., $\downarrow t \stackrel{def}{=} \{t' \in \mathbf{T} \mid t' \lesssim t\}$. For any $s \in \mathbf{S}_\mathcal{T}$, the three following equalities hold:*

$$\{t \in \mathbf{T} \mid s \in \boldsymbol{\alpha}(t)\} = \quad \downarrow \boldsymbol{\gamma}(s) \tag{1}$$
$$\{t \in \mathbf{T} \mid s \in \boldsymbol{\alpha_1}(t)\} = \quad \downarrow \boldsymbol{\gamma_1}(s) \tag{2}$$
$$\{t \in \mathbf{T} \mid s \in \boldsymbol{\alpha_0}(t)\} = \quad \downarrow \boldsymbol{\gamma_0}(s) \tag{3}$$

Proof. Let $s \in \boldsymbol{S_\mathcal{T}}$.
$\subseteq$: Let $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha}(t)\}$. $s \in \boldsymbol{\alpha}(t)$, by definition 3.8, $t \lesssim \boldsymbol{\gamma}(s)$, so $t \in \downarrow \boldsymbol{\gamma}(s)$.
$\supseteq$: Let $t \in \downarrow \boldsymbol{\gamma}(s)$. $t \lesssim \boldsymbol{\gamma}(s)$, and by definition 3.8, $s \in \boldsymbol{\alpha}(t)$, so $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha}(t)\}$.
$\subseteq$: Let $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha_1}(t)\}$. $s \in \boldsymbol{\alpha_1}(t)$, by definition 3.9, $t \lesssim \boldsymbol{\gamma_1}(s)$, so $t \in \downarrow \boldsymbol{\gamma_1}(s)$.
$\supseteq$: Let $t \in \downarrow \boldsymbol{\gamma_1}(s)$. $t \lesssim \boldsymbol{\gamma_1}(s)$, and by definition 3.9, $s \in \boldsymbol{\alpha_1}(t)$, so $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha_1}(t)\}$.
$\subseteq$: Let $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha_0}(t)\}$. $s \in \boldsymbol{\alpha_0}(t)$, by definition 3.10, $t \lesssim \boldsymbol{\gamma_0}(s)$, so $t \in \downarrow \boldsymbol{\gamma}(s)$.
$\supseteq$: Let $t \in \downarrow \boldsymbol{\gamma_0}(s)$. $t \lesssim \boldsymbol{\gamma}(s)$, and by definition 3.10, $s \in \boldsymbol{\alpha_0}(t)$, so $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha_0}(t)\}$. □

As a direct consequence, as units get more and more specific (i.e., as we go down the hierarchy of PUTs), the set of ASlots (resp1. OblASlots, resp2. ProASlots) may only increase. We say that ASlots (resp1. OblASlots, resp2. ProASlots) are inherited:

Proposition 3.4. *ASlots, OblASlots and ProASlots are inherited, i.e., $\forall t_x, t_y \in \mathbf{T}$ such that $t_x \lesssim t_y$,*

$$\boldsymbol{\alpha}(t_y) \quad \subseteq \boldsymbol{\alpha}(t_x) \tag{4}$$
$$\boldsymbol{\alpha_1}(t_y) \quad \subseteq \boldsymbol{\alpha_1}(t_x) \tag{5}$$
$$\boldsymbol{\alpha_0}(t_y) \quad \subseteq \boldsymbol{\alpha_0}(t_x) \tag{6}$$

Moreover, if $t_x \simeq t_y$, then $\boldsymbol{\alpha}(t_y) = \boldsymbol{\alpha}(t_x)$, $\boldsymbol{\alpha_1}(t_y) = \boldsymbol{\alpha_1}(t_x)$, and $\boldsymbol{\alpha_0}(t_y) = \boldsymbol{\alpha_0}(t_x)$.

Proof. Let $t_x, t_y \in \boldsymbol{T}$ such that $t_x \lesssim t_y$.
Let $s \in \boldsymbol{\alpha}(t_y)$. Definition 3.8 implies $t_y \lesssim \boldsymbol{\gamma}(s)$. So $t_x \lesssim t_y \lesssim \boldsymbol{\gamma}(s)$ and $s \in \boldsymbol{\alpha}(t_x)$.
Let $s \in \boldsymbol{\alpha_1}(t_y)$. Definition 3.9 implies $t_y \lesssim \boldsymbol{\gamma_1}(s)$. So $t_x \lesssim t_y \lesssim \boldsymbol{\gamma_1}(s)$ and $s \in \boldsymbol{\alpha_1}(t_x)$.
Let $s \in \boldsymbol{\alpha_0}(t_y)$. Definition 3.10 implies $t_y \lesssim \boldsymbol{\gamma_0}(s)$. So $t_x \lesssim t_y \lesssim \boldsymbol{\gamma_0}(s)$ and $s \in \boldsymbol{\alpha_0}(t_x)$.
The second results are obtained using twice these first results. □

Second, for every ASymbol the obligat (resp. prohibet) is lower than the radix. So the set of OblASlots (resp. ProASlot) is always a subset of the set of ASlots:

Proposition 3.5. *For any PUT* $t \in \mathbf{T}$*,*

$$\boldsymbol{\alpha_1}(t) \subseteq \boldsymbol{\alpha}(t) \tag{7}$$
$$\boldsymbol{\alpha_0}(t) \subseteq \boldsymbol{\alpha}(t) \tag{8}$$

Proof. Let $t \in \boldsymbol{T}$, and $s \in \boldsymbol{\alpha_1}(t)$.
From definition 3.9, $t \lesssim \boldsymbol{\gamma_1}(s)$.
Then as $\boldsymbol{\gamma_1}(s) \lesssim \boldsymbol{\gamma}(s)$ from proposition 3.1, we know that $t \lesssim \boldsymbol{\gamma}(s)$.
So by definition 3.8, $s \in \boldsymbol{\alpha}(t)$ and $\boldsymbol{\alpha_1}(t) \subseteq \boldsymbol{\alpha}(t)$.
The proof for $\boldsymbol{\alpha_0}$ is the same with replacing $\boldsymbol{\gamma_1}$ by $\boldsymbol{\gamma_0}$, definition 3.9 by definition 3.10, and $\boldsymbol{\alpha_1}$ by $\boldsymbol{\alpha_0}$. □

Finally, as $\boldsymbol{\gamma}$ (resp1. $\boldsymbol{\gamma_1}$, resp2. $\boldsymbol{\gamma_0}$) is a mapping, every ASymbol has a radix (resp1. obligat, resp2. prohibet). Any minimal element of $\boldsymbol{T}$ will inherit all of the ASlots (resp1. OblASlots, resp2. ProASlots):

Proposition 3.6. $\forall t \in \perp^{\sim}$,

$$\boldsymbol{\alpha}(t) = \boldsymbol{\alpha_1}(t) = \boldsymbol{\alpha_0}(t) = \mathbf{S}_{\mathcal{T}}$$

Proof. As $\perp$ is a minimal element in $\boldsymbol{T}$, then $\forall s \in \boldsymbol{S}_{\mathcal{T}}, \perp \lesssim \boldsymbol{\gamma}(s)$. So $s \in \boldsymbol{\alpha}(\perp)$. Thus $\boldsymbol{\alpha}(\perp) = \boldsymbol{S}_{\mathcal{T}}$.
Now, for all $t \in \perp^{\sim}$ we know that $t \lesssim \perp$, and using proposition 3.4 $\boldsymbol{S}_{\mathcal{T}} \subseteq \boldsymbol{\alpha}(t)$. So $\boldsymbol{\alpha}(t) = \boldsymbol{S}_{\mathcal{T}}$.
The proof for $\boldsymbol{\alpha_1}$ (resp. $\boldsymbol{\alpha_0}$) is the same with replacing $\boldsymbol{\gamma}$ by $\boldsymbol{\gamma_1}$ (resp. $\boldsymbol{\gamma_0}$), and $\boldsymbol{\alpha}$ by $\boldsymbol{\alpha_1}$ (resp. $\boldsymbol{\alpha_0}$). □

3.3 Optional ASlots

Let $t \in \boldsymbol{T}$ be a PUT. The set of Optional Actant Slots (OptASlots) of t is the set of ASlots which are neither obligatory nor prohibited.

Definition 3.11 (OptASlots of a PUT)**.** The set of *Optional Actant Slots (OptASlots)* of PUTs is defined by a mapping $\boldsymbol{\alpha_?}$ from $\boldsymbol{T}$ to $2^{\boldsymbol{S_T}}$, that associates every PUT $t \in \boldsymbol{T}$ with the set $\boldsymbol{\alpha_?} = \boldsymbol{\alpha} - \boldsymbol{\alpha_1} - \boldsymbol{\alpha_0}$.

The following property thus holds:

Due to proposition 3.3, we know that any PUT with OptASlot s is in lower set $\downarrow\boldsymbol{\gamma}(obj)$ and is not in lower sets $\downarrow\boldsymbol{\gamma_1}(obj)$ nor $\downarrow\boldsymbol{\gamma_0}(obj)$. So the following proposition holds:

Proposition 3.7. *For any* $s \in \mathbf{S}_{\mathcal{T}}$*,*

$$\{t \in \mathbf{T} \mid s \in \boldsymbol{\alpha_?}(t)\} = \downarrow\boldsymbol{\gamma}(s) - \downarrow\boldsymbol{\gamma_1}(s) - \downarrow\boldsymbol{\gamma_0}(s) = \{t \in \mathbf{T} \mid t \lesssim \boldsymbol{\gamma}(s) \text{ and } t \not\lesssim \boldsymbol{\gamma_1}(s) \text{ and } t \not\lesssim \boldsymbol{\gamma_0}(s)\} \tag{9}$$

Proof. Let $s \in \boldsymbol{S_T}$.
$\subseteq$: Let $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha_?}(t)\}$. $s \in \boldsymbol{\alpha_?}(t)$.
By definition 3.11, $s \in \boldsymbol{\alpha}(t) - \boldsymbol{\alpha_1}(t) - \boldsymbol{\alpha_0}(t)$.
By definitions 3.8, 3.9 and 3.10, $s \lesssim \boldsymbol{\gamma}(s)$ and $t \not\lesssim \boldsymbol{\gamma_1}(s)$ and $t \not\lesssim \boldsymbol{\gamma_0}(s)$.
So $t \in \downarrow\boldsymbol{\gamma}(s) - \downarrow\boldsymbol{\gamma_1}(s) - \downarrow\boldsymbol{\gamma_0}(s)$.
$\supseteq$: Let $t \in \downarrow\boldsymbol{\gamma}(s) - \downarrow\boldsymbol{\gamma_1}(s) - \downarrow\boldsymbol{\gamma_0}(s)$.
$s \lesssim \boldsymbol{\gamma}(s)$ and $t \not\lesssim \boldsymbol{\gamma_1}(s)$ and $t \not\lesssim \boldsymbol{\gamma_0}(s)$.
By definition 3.8, 3.9 and 3.10, $s \in \boldsymbol{\alpha}(t)$ and $s \notin \boldsymbol{\alpha_1}(t)$and $s \notin \boldsymbol{\alpha_0}(t)$.
So $s \in \boldsymbol{\alpha_?}(t)$, and $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha_?}(t)\}$. □

Thus in the hierarchy of PUTs, an ASlot s is introduced by $\boldsymbol{\gamma}(s)$ and first defines a OblASlot until s becomes more specific than $\boldsymbol{\gamma_1}(s)$ (resp. $\boldsymbol{\gamma_0}(s)$. If that happens, then ASlot s becomes obligatory (resp. prohibited). Now from propositions 3.4, one concludes that two equivalent PUTs share the same set of OptASlots:

Proposition 3.8. $\forall t, t' \in \mathbf{T}$ *such that* $t \simeq t'$, $\boldsymbol{\alpha_?}(t) = \boldsymbol{\alpha_?}(t')$.

Proof. From definition 3.8, $\boldsymbol{\alpha_?}(t) = \boldsymbol{\alpha}(t) - \boldsymbol{\alpha_1}(t) - \boldsymbol{\alpha_0}(t)$.
Moreover, from propositions 3.4, $\boldsymbol{\alpha}(t') = \boldsymbol{\alpha}(t)$, $\boldsymbol{\alpha_1}(t') = \boldsymbol{\alpha_1}(t)$ and $\boldsymbol{\alpha_0}(t') = \boldsymbol{\alpha_0}(t)$.
So $\boldsymbol{\alpha}(t') = \boldsymbol{\alpha}(t)$. □

Finally, from propositions 3.6, one concludes that any absurd PUT has no OptASlot:

Proposition 3.9. $\forall t \in \perp^{\sim}, \boldsymbol{\alpha_?}(t) = \varnothing$.

Proof. Let $t \in \perp^{\sim}$. From definition 3.11 and proposition 3.6, $\boldsymbol{\alpha_?}(t) = \boldsymbol{S_T} - \boldsymbol{S_T} - \boldsymbol{S_T} = \varnothing$. □

3.4 PUT Signatures

For any PUT, not any unit may fill one of its specific ASlot. For instance, any unit that fills ASlot *obj* of a unit with type /move\ should be of type /object\, and absolutely not of type /idea\ nor /eat\. *Signatures* enable the assertion of what type fillers of ASlots are.

As PUTs get more and more specific, the signature of a given common ASlot may only become more and more specific. For instance, any unit that fills ASlot *obj* of a unit with type /move\ should be of type /object\, but only units with type /weighing object\ may fill ASlots *obj* of a unit with type /fall\.

Definition 3.12 (Signatures of PUTs)**.** The set of signatures of PUTs $\{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}}$ is a set of functions from $\boldsymbol{S}_\mathcal{T}$ to $2^{\boldsymbol{T}}$. For every PUT t, $\boldsymbol{\varsigma}_t$ is a function with $domain(\boldsymbol{\varsigma}_t) = \boldsymbol{\alpha}(t)$ that associates to each ASlot s of t a set of PUT $\boldsymbol{\varsigma}_t(s)$ that characterize the type units that may fill this slot should be. The set of signatures $\{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}}$ must be such that for all $t_1, t_2 \in \boldsymbol{T}$, and $s \in \boldsymbol{S}_\mathcal{T}$ such that $t_1 \lesssim t_2$ and $s \in \boldsymbol{\alpha}(t_2)$, $\forall t'_2 \in \boldsymbol{\varsigma}_{t_2}(s), \exists t'_1 \in \boldsymbol{\varsigma}_{t_1}(s) : t'_1 \lesssim t'_2$.

The above definition is complex due to the fact that important notions will be introduced in next section. We will see that $2^{\boldsymbol{T}}$ is the set of so-called Conjunctive Unit Types (CUTs), and thus the signature $\boldsymbol{\varsigma}_t$ of t is a CUT. Moreover, when the pre-order relation $\overset{\cap}{\lesssim}$ over the set of CUTs will be introduced, we will see that $\forall t'_2 \in \boldsymbol{\varsigma}_{t_2}(s), \exists t'_1 \in \boldsymbol{\varsigma}_{t_1}(s) : t'_1 \lesssim t'_2$ implies $\boldsymbol{\varsigma}_{t_1}(s) \overset{\cap}{\lesssim} \boldsymbol{\varsigma}_{t_2}(s)$. Thus the set of signatures $\{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}}$ must be such that for all $t_1, t_2 \in \boldsymbol{T}$, and $s \in \boldsymbol{S}_\mathcal{T}$ such that $s \in \boldsymbol{\alpha}(t_2)$,

$$t_1 \lesssim t_2 \Rightarrow \boldsymbol{\varsigma}_{t_1}(s) \overset{\cap}{\lesssim} \boldsymbol{\varsigma}_{t_2}(s) \tag{10}$$

4 Conjunctive Unit Types (CUTs)

In the Meaning-Text Theory (MTT), a unit may actually have multiple types. For instance, a unit may have a have a lexical unit type and a set of grammatical unit types for instance, like $\{def, sing, \text{CAT}\}$ for *"the cat"*.

In section 4.1 we hence introduce the set of Conjunctive Unit Type (CUT) as the powerset of $\boldsymbol{T}$, and will define the set of asserted absurd CUT set, which is a set of combinations of PUTs that may not have instances.

Any CUT may also be seen both as a concept and as a n-ary relation, inheriting the actantial structure of the PUTs it is composed of. Section 4.2 introduces slots and signatures for CUTs.

Then, section 4.3 introduces a specialization pre-order over CUTs, which is build from the pre-order over constituent PUTs, asserted absurd CUTs and absurd signatures. A natural definition of absurd CUTs is then given.

Finally, a definition of the hierarchy of unit types is introduced in 4.4.

4.1 Definition of CUTs

A unit type may consist of several conjoint PUTs. We hence introduce the set of possible Conjunctive Unit Type (CUT) which is the powerset of $\boldsymbol{T}$, i.e., the set of all subsets of $\boldsymbol{T}$:

Definition 4.1 (CUT set over $\boldsymbol{T}$)**.** The *CUT set* over $\boldsymbol{T}$ is the set of all subsets of $\boldsymbol{T}$, i.e., $\boldsymbol{T}^{\cap} \stackrel{\text{def}}{=} 2^{\boldsymbol{T}}$. Every element in $\boldsymbol{T}^{\cap}$ is denoted both *CUT* and by the set itself. $\top^{\cap} = \{\top\}$ is denoted the *prime universal CUT*, and $\bot^{\cap} = \{\bot\}$ the *prime absurd CUT*. Any singleton CUT $\{t\}$ is said to be primitive, i.e., a primitive CUT.

Some CUTs such as $\{def, indef\}$ are said to be absurd. This means there is no unit that has both types def and $indef$. Asserted absurd CUT set is defined as follows:

Definition 4.2 (Asserted absurd CUT set)**.** The set of *asserted absurd CUTs* is a set of CUTs that is denoted by $\bot^{\sqcap}_{A}$. And $\bot^{\sqcap}_{A} \subseteq \boldsymbol{T}^{\cap}$.

We will see that by definition the prime absurd CUT $\bot^{\cap}$ is absurd, and that for all $s \in \boldsymbol{S}_{\mathcal{T}}$, $\{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\}$ is absurd.

4.2 CUT Slots and Signatures

The actantial structure of PUTs, i.e., ASlots, OblASlots, ProASlots, OptASlots and signatures are naturally extended to CUTs.

For instance, consider a CUT $t^{\cap} = \{/\text{move}\backslash, /\text{quick}\backslash\}$. $t^{\cap}$ contains the PUT $/\text{move}\backslash$, and as $obj \in \boldsymbol{\alpha}(/\text{move}\backslash)$, obj corresponds to a participant of SIT(move). As SIT(move,quick) is a specialization of SIT(move), we consider that obj is also a ASlot of $t^{\cap}$:

Definition 4.3 (ASlots of CUTs, and valency of a CUT)**.** The set of *ASlots of CUTs* is defined through a mapping $\boldsymbol{\alpha}^{\cap}$ from $\boldsymbol{T}^{\cap}$ to $2^{\boldsymbol{S}_{\mathcal{T}}}$ Such that $\forall t^{\cap} \in \boldsymbol{T}^{\cap}, \boldsymbol{\alpha}^{\cap}(t^{\cap}) \stackrel{\text{def}}{=} \bigcup_{t \in t^{\cap}} \boldsymbol{\alpha}(t)$. The number of ASlots of a CUT $t^{\cap}$ is denoted the *valency* of $t^{\cap}$, i.e., $valency(t^{\cap}) \stackrel{\text{def}}{=} |\boldsymbol{\alpha}^{\cap}(t^{\cap})|$.

Definitions of OblASlots and ProASlots of PUTs are also naturally extended to CUTs.

Definition 4.4 (OblASlots of a CUTs)**.** The set of *OblASlots of CUTs* is defined through a mapping $\boldsymbol{\alpha}^{\cap}_{\boldsymbol{1}}$ from $\boldsymbol{T}^{\cap}$ to $2^{\boldsymbol{S}_{\mathcal{T}}}$ Such that $\forall t^{\cap} \in \boldsymbol{T}^{\cap}, \boldsymbol{\alpha}^{\cap}_{\boldsymbol{1}}(t^{\cap}) \stackrel{\text{def}}{=} \bigcup_{t \in t^{\cap}} \boldsymbol{\alpha_1}(t)$.

Definition 4.5 (ProASlots of a CUTs)**.** The set of *ProASlots of CUTs* is defined through a mapping $\boldsymbol{\alpha_0^\cap}$ from $\boldsymbol{T^\cap}$ to $2^{\boldsymbol{S_\mathcal{T}}}$ Such that $\forall t^\cap \in \boldsymbol{T^\cap}, \boldsymbol{\alpha_0^\cap}(t^\cap) \stackrel{\text{def}}{=} \bigcup_{t \in t^\cap} \boldsymbol{\alpha_0}(t)$.

The inheritance property 3.4 is preserved for CUTs.

Proposition 4.1. *For any CUT* $t^\cap \in \mathbf{T}^\cap$,

$$\boldsymbol{\alpha_1^\cap}(t^\cap) \subseteq \boldsymbol{\alpha}^\cap(t^\cap) \tag{11}$$
$$\boldsymbol{\alpha_0^\cap}(t^\cap) \subseteq \boldsymbol{\alpha}^\cap(t^\cap) \tag{12}$$

Proof. Let $t^\cap \in \boldsymbol{T^\cap}$ and $s \in \boldsymbol{\alpha_1^\cap}(t^\cap)$ (resp. $\boldsymbol{\alpha_0^\cap}(t^\cap)$).
By definition 4.4 (resp. 4.5), $\exists t \in t^\cap, s \in \boldsymbol{\alpha_1}(t)$ (resp. $\boldsymbol{\alpha_0}(t)$).
From proposition 3.5, $s \in \boldsymbol{\alpha}(t)$.
By definition 4.3, $s \in \boldsymbol{\alpha}^\cap(t^\cap)$. □

Finally, similarly to PUTs, the OptASlots of a CUT is defined as the difference between the set of ASlots and the set of OblASlots:

Definition 4.6 (OptASlots of a CUT)**.** The set of *OptASlots of CUTs* is defined through a mapping $\boldsymbol{\alpha_?^\cap}$ from $\boldsymbol{T^\cap}$ to $2^{\boldsymbol{S_\mathcal{T}}}$ Such that $\forall t^\cap \in \boldsymbol{T^\cap}, \boldsymbol{\alpha_?^\cap}(t^\cap) \stackrel{\text{def}}{=} \boldsymbol{\alpha}^\cap(t^\cap) - \boldsymbol{\alpha_1^\cap}(t^\cap) - \boldsymbol{\alpha_?^\cap}(t^\cap)$.
The number of OptASlots of a CUT $t^\cap$ is denoted the *valency* of $t^\cap$, i.e., $valency(t^\cap) \stackrel{\text{def}}{=} |\boldsymbol{\alpha_?^\cap}(t^\cap)|$.

Signatures are also naturally extended to CUTs, but for the ASlot *obj* for instance, we must take the union of signatures of PUTs constituents over a more complex set: the set of PUTs that have the ASlot *obj*: $\{t \in t^\cap \mid obj \in \boldsymbol{\alpha}(t)\}$:

Definition 4.7 (Signature of a CUT)**.** The set of signatures of CUTs $\{\boldsymbol{\varsigma_{t^\cap}^\cap}\}_{t^\cap \in \boldsymbol{T^\cap}}$ is a set of functions from $\boldsymbol{S_\mathcal{T}}$ to $\boldsymbol{T^\cap}$. For every CUT $t^\cap$, $\boldsymbol{\varsigma_{t^\cap}^\cap}$ is a function with $domain(\boldsymbol{\varsigma_{t^\cap}^\cap}) = \boldsymbol{\alpha}^\cap(t^\cap)$ such that for all s in $\boldsymbol{\alpha}^\cap(t^\cap)$, $\boldsymbol{\varsigma_{t^\cap}^\cap}(s) \stackrel{\text{def}}{=} \bigcup_{t \in t^\cap | s \in \boldsymbol{\alpha}(t)} \boldsymbol{\varsigma_t}(s)$.

Straightforwardly, the only ASlots of a singleton CUT are the ASlots of its single PUT element:

Proposition 4.2. *For all* $t \in \mathbf{T}$*, all of the following is true:*

- $\boldsymbol{\alpha}^\cap(\{t\}) = \boldsymbol{\alpha}(t)$.
- $\boldsymbol{\alpha_1^\cap}(\{t\}) = \boldsymbol{\alpha_1}(t)$.
- $\boldsymbol{\alpha_0^\cap}(\{t\}) = \boldsymbol{\alpha_0}(t)$.
- $\boldsymbol{\alpha_?^\cap}(\{t\}) = \boldsymbol{\alpha_?}(t)$.
- *for all* $s \in \boldsymbol{\alpha}(t)$, $\boldsymbol{\varsigma_{\{t\}}^\cap}(s) = \boldsymbol{\varsigma_t}(s)$

Proof. Let $t \in \boldsymbol{T}$.
1) $\boldsymbol{\alpha}^\cap(\{t\}) = \bigcup_{t \in \{t\}} \boldsymbol{\alpha}(t) = \boldsymbol{\alpha}(t)$;
2) $\boldsymbol{\alpha_1^\cap}(\{t\}) = \bigcup_{t \in \{t\}} \boldsymbol{\alpha_1}(t) = \boldsymbol{\alpha_1}(t)$;
3) $\boldsymbol{\alpha_0^\cap}(\{t\}) = \bigcup_{t \in \{t\}} \boldsymbol{\alpha_0}(t) = \boldsymbol{\alpha_0}(t)$;
4) $\boldsymbol{\alpha_?^\cap}(\{t\}) = \boldsymbol{\alpha}^\cap(\{t\}) - \boldsymbol{\alpha_1^\cap}(\{t\}) - \boldsymbol{\alpha_0^\cap}(\{t\}) = \boldsymbol{\alpha}(t) - \boldsymbol{\alpha_1}(t) - \boldsymbol{\alpha_0}(t) = \boldsymbol{\alpha_?}(t)$.
5) let $s \in \boldsymbol{\alpha}(t)$. $\{t \in \{t\} \mid s \in \boldsymbol{\alpha}(t)\} = \{t\}$, so $\boldsymbol{\varsigma_{\{t\}}^\cap}(s) = \bigcup_{t \in \{t\}} \boldsymbol{\varsigma_t}(s) = \boldsymbol{\varsigma_t}(s)$. □

4.3 Pre-order over CUTs

In this section we define a pre-order over $\boldsymbol{T}^{\cap}$ which models a specialization relation, e.g., $\{/\text{happy}^{\backslash}, /\text{cat}^{\backslash}\} \overset{\cap}{\lesssim} \{/\text{hasMood}^{\backslash}, /\text{animal}^{\backslash}\}$ means that $\{/\text{happy}^{\backslash}, /\text{cat}^{\backslash}\}$ is more specific (semantically) than $\{/\text{hasMood}^{\backslash}, /\text{animal}^{\backslash}\}$.

Let us first introduce the definition of an iterative construction process of a comparisons set over $\boldsymbol{T}^{\cap}$, and we will describe the intuitive meaning of every element of this definition just after:

Definition 4.8 (Comparisons set over $\boldsymbol{T}^{\cap}$)**.** Let $(\boldsymbol{C}_n^{\cap})_{n\in\mathbb{N}}$ be a sequence of comparisons sets over $\boldsymbol{T}^{\cap}$, i.e., for all $n \in \mathbb{N}$, $\boldsymbol{C}_n^{\cap} \subseteq \boldsymbol{T}^{\cap 2}$, defined by

- for $i = 0$, $\boldsymbol{C}_0^{\cap} \overset{\text{def}}{=} \boldsymbol{C}_{\lesssim}^{\cap} \cup \boldsymbol{C}_{\top}^{\cap} \cup \boldsymbol{C}_{\bot}^{\cap}$, where:

$$\boldsymbol{C}_{\lesssim}^{\cap} \overset{\text{def}}{=} \{(t_y^{\cap}, t_x^{\cap}) \in \boldsymbol{T}^{\cap 2} \mid \forall t_y \in t_y^{\cap}, \exists t_x \in t_x^{\cap} : t_x \lesssim t_y\} \tag{13}$$

$$\boldsymbol{C}_{\top}^{\cap} \overset{\text{def}}{=} \{(\top^{\cap}, \varnothing)\} \tag{14}$$

$$\boldsymbol{C}_{\mathbf{\Gamma}}^{\cap} \overset{\text{def}}{=} \{(\bot^{\cap}, \{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\})\}_{s\in\boldsymbol{s}_{\mathcal{T}}} \tag{15}$$

$$\boldsymbol{C}_{\bot}^{\cap} \overset{\text{def}}{=} \{(\bot^{\cap}, t^{\cap}) \in \boldsymbol{T}^{\cap 2} \mid t^{\cap} \in \bot_A^{\sqcap}\} \tag{16}$$

- for all $i > 0$, $\boldsymbol{C}_i^{\cap} \overset{\text{def}}{=} \boldsymbol{C}_{i-1}^{\cap} \cup \boldsymbol{C}_{\boldsymbol{\varsigma}\,i}^{\cap} \cup \boldsymbol{C}_{+i}^{\cap}$ where:

$$\boldsymbol{C}_{\boldsymbol{\varsigma}\,i}^{\cap} \overset{\text{def}}{=} \{(\bot^{\cap}, t^{\cap}) \in \boldsymbol{T}^{\cap 2} \mid \exists s \in \boldsymbol{\alpha}^{\cap}(t^{\cap}), (\bot^{\cap}, \boldsymbol{\varsigma}_{t^{\cap}}^{\cap}(s)) \in \boldsymbol{C}_{i-1}^{\cap}\} \tag{17}$$

$$\boldsymbol{C}_{+i}^{\cap} \overset{\text{def}}{=} \{(t_z^{\cap}, t_x^{\cap}) \in \boldsymbol{T}^{\cap 2} \mid \exists t_y^{\cap} \in \boldsymbol{T}^{\cap}, (t_z^{\cap}, t_y^{\cap}) \in \boldsymbol{C}_{i-1}^{\cap} \text{ and } (t_y^{\cap}, t_x^{\cap}) \in \boldsymbol{C}_{i-1}^{\cap}\} \tag{18}$$

The sequence $(\boldsymbol{C}_n^{\cap})$ is a bounded monotonic increasing sequence, i.e., for all $n \in \mathbb{N}$, $\boldsymbol{C}_n^{\cap} \subseteq \boldsymbol{C}_{n+1}^{\cap} \subseteq \boldsymbol{T}^{\cap 2}$, so it is convergent. The least upper bound of the sequence $(\boldsymbol{C}_n^{\cap})_{n\in\mathbb{N}}$ is denoted $\boldsymbol{C}^{\cap}$: the *comparisons set over* $\mathbf{T}^{\cap}$.

The intuitive meaning of the iterative construction process is the following:

- $\boldsymbol{C}_{\lesssim}^{\cap}$ represents the natural extension of each pre-order over a set to a pre-order over its powerset;
- $\boldsymbol{C}_{\top}^{\cap}$ is introduced to flatten the top of the powerset so that $\top^{\cap}$ is the most generic CUT;
- $\boldsymbol{C}_{\mathbf{\Gamma}}^{\cap}$ represents the fact that for every ASymbol the obligat and the prohibet are incompatible;
- $\boldsymbol{C}_{\bot}^{\cap}$ is introduced to flatten the bottom of the powerset so that any asserted absurd CUT is considered as a minimal CUT (a most specific CUT);
- $\boldsymbol{C}_{\boldsymbol{\varsigma}\,i}^{\cap}$ represents the fact that if a signature of a CUT for a given ASlot is absurd, then that CUT is absurd;
- $\boldsymbol{C}_{+i}^{\cap}$ transitively closes the set of comparisons $\boldsymbol{C}^{\cap}$.

It is a well know result that $\boldsymbol{C}_{\lesssim}^{\cap}$ defines a pre-order over $2^{\boldsymbol{T}}$ (i.e., it is reflexive and transitive), let us recall how this result is obtained:

Proposition 4.3. $\mathbf{C}_{\lesssim}^{\cap}$ *is reflexive and transitive.*

Proof. Reflexivity: Let $t^\cap \in \boldsymbol{T}^\cap$.
$\forall t \in t^\cap, \exists t'(=t) \in t^\cap : t' \lesssim t$.
So $(t^\cap, t^\cap) \in \boldsymbol{C}^\cap_{\lesssim}$.
Transitivity: Let $t^\cap_x, t^\cap_y, t^\cap_z \in \boldsymbol{T}^\cap$ such that $(t^\cap_z, t^\cap_y) \in \boldsymbol{C}^\cap_{\lesssim}$, and $(t^\cap_y, t^\cap_x) \in \boldsymbol{C}^\cap_{\lesssim}$.
$\forall t_z \in t^\cap_z, \exists t_y \in t^\cap_y : t_y \lesssim t_z$, and $\forall t'_y \in t^\cap_y, \exists t_x \in t^\cap_x : t_x \lesssim t'_y$.
Let $t'_y = t_y$, then $\forall t_z \in t^\cap_z, \exists t_x \in t^\cap_x : t_x \lesssim t_z$,
and thus $(t^\cap_z, t^\cap_x) \in \boldsymbol{C}^\cap_{\lesssim}$. □

Now from that result and the construction process of $\boldsymbol{C}^\cap$, we may prove the reflexivity and the transitivity of $\boldsymbol{C}^\cap$:

Proposition 4.4. $\mathbf{C}^\cap$ *is reflexive and transitive.*

Proof. As $\boldsymbol{C}^\cap_{\lesssim} \subseteq \boldsymbol{C}^\cap$ and $\boldsymbol{C}^\cap_{\lesssim}$ is reflexive, then $\boldsymbol{C}^\cap$ is reflexive.
Transitivity is obtained from $\boldsymbol{C}^\cap_+$ in the construction process of $\boldsymbol{C}^\cap$. □

Now that we know $\boldsymbol{C}^\cap$ is reflexive and transitive, it defines a pre-order relation over $\boldsymbol{T}^\cap$:

Definition 4.9 (Pre-order relation over $\boldsymbol{T}^\cap$)**.** $\boldsymbol{C}^\cap$ is a binary relation over $\boldsymbol{T}^\cap$which is transitive and reflexive. So it defines a pre-order relation $\overset{\cap}{\lesssim}$ over $\boldsymbol{T}^\cap$, i.e., $t^\cap_x \overset{\cap}{\lesssim} t^\cap_y$ if and only if $(t^\cap_y, t^\cap_x) \in \boldsymbol{C}^\cap$.

The following proposition underlines sufficient conditions so that a CUT is more specific than another CUT. Each of these conditions is the expression of one of the intuitive meaning of the elements in the construction process of the comparisons set over $\boldsymbol{T}^\cap$:

Proposition 4.5. *The pre-order relation* $\overset{\cap}{\lesssim}$ *is such that:*

- ***Extension of*** $\lesssim$ *For all* $t^\cap_x, t^\cap_y \in \mathbf{T}^\cap$*, if* $\forall t_y \in t^\cap_y, \exists t_x \in t^\cap_x : t_x \lesssim t_y$*, then* $t^\cap_x \overset{\cap}{\lesssim} t^\cap_y$*;*
- ***Top CUT*** $\varnothing \overset{\cap}{\lesssim} \top^\cap$*;*
- ***Absurd ASlots*** *For all* $t^\cap \in \mathbf{T}^\cap$*, if* $\boldsymbol{\alpha}^\cap_{\boldsymbol{1}}(t^\cap) \cap \boldsymbol{\alpha}^\cap_{\boldsymbol{0}}(t^\cap) \neq \varnothing$*, then* $t^\cap \overset{\cap}{\lesssim} \bot^\cap$*;*
- ***Asserted absurd types*** *For all* $t^\cap \in \bot^\sqcap_A$*,* $t^\cap \overset{\cap}{\lesssim} \bot^\cap$*;*
- ***Absurd signatures*** *For all* $t^\cap \in \mathbf{T}^\cap$*, if there exists* $s \in \boldsymbol{\alpha}^\cap(t^\cap)$ *such that* $\boldsymbol{\varsigma}^\cap_{t^\cap}(s) \overset{\cap}{\lesssim} \bot^\cap$*, then* $t^\cap \overset{\cap}{\lesssim} \bot^\cap$*;*

Proof. Extension of $\lesssim$*:* For all $t^\cap_x, t^\cap_y \in \boldsymbol{T}^\cap$, if $\forall t_y \in t^\cap_y, \exists t_x \in t^\cap_x : t_x \lesssim t_y$, then $(t^\cap_y, t^\cap_x) \in \boldsymbol{C}^\cap_{\lesssim} \subseteq \boldsymbol{C}^\cap$, so $t^\cap_x \overset{\cap}{\lesssim} t^\cap_y$.
Top CUT: We know that $(\top^\cap, \varnothing) \in \boldsymbol{C}^\cap$, so $\varnothing \overset{\cap}{\lesssim} \top^\cap$.
Asserted ASlots: For all $t^\cap \in \bot^\sqcap_A$, $(\bot^\cap, t^\cap) \in \boldsymbol{C}^\cap_\bot \subseteq \boldsymbol{C}^\cap$, so $t^\cap \overset{\cap}{\lesssim} \bot^\cap$.
Asserted absurd types: Let $t^\cap \in \boldsymbol{T}^\cap$ such that $s \in \boldsymbol{\alpha}^\cap_{\boldsymbol{1}}(t^\cap) \cap \boldsymbol{\alpha}^\cap_{\boldsymbol{0}}(t^\cap)$. From definitions 4.4 and 4.5, $\exists t_1 \in t^\cap : s \in \boldsymbol{\alpha_1}(t)$, and $\exists t_0 \in t^\cap : s \in \boldsymbol{\alpha_0}(t)$. So $t^\cap \overset{\cap}{\lesssim} \{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\} \overset{\cap}{\lesssim} \bot^\cap$.
Absurd signatures: For all $t^\cap \in \boldsymbol{T}^\cap$, if there exists $s \in \boldsymbol{\alpha}^\cap(t^\cap)$ such that $\boldsymbol{\varsigma}^\cap_{t^\cap}(s) \overset{\cap}{\lesssim} \bot^\cap$, then $(\bot^\cap, \boldsymbol{\varsigma}^\cap_{t^\cap}(s)) \in \boldsymbol{C}^\cap$. There exists $n \in \mathbb{N}$ such that $(\bot^\cap, \boldsymbol{\varsigma}^\cap_{t^\cap}(s)) \in \boldsymbol{C}^\cap_n$, and thus for $n+1$, $(\bot^\cap, t^\cap) \in \boldsymbol{C}^\cap_{\boldsymbol{\varsigma}\, n+1} \subseteq \boldsymbol{C}^\cap$. So $t^\cap \overset{\cap}{\lesssim} \bot^\cap$. □

In the mathematical theory of categories, pre-ordered sets has category denoted **Ord** with monotonic functions as homomorphisms. In our case here, we have two pre-ordered sets: the powerset of $\boldsymbol{T}^\cap$ with the inclusion relation $(\boldsymbol{T}^\cap, \subseteq)$ and the powerset of $\boldsymbol{T}^\cap$ with the specialization relation $(\boldsymbol{T}^\cap, \overset{\cap}{\lesssim})$. The morphism f that maps one onto the other and such that $f(t^\cap) = t^\cap$ is anti-monotonic:

Proposition 4.6. *The function* $f : (\mathbf{T}^\cap, \subseteq) \to (\mathbf{T}^\cap, \overset{\cap}{\lesssim})$ *such that* $f(t^\cap) = t^\cap$ *is an anti-monotonic pre-order homomorphism, i.e., for all* $t^\cap_x, t^\cap_y \in \mathbf{T}^\cap$, $t^\cap_x \subseteq t^\cap_y \Rightarrow t^\cap_y \overset{\cap}{\lesssim} t^\cap_x$.

Proof. Let $t^\cap_x, t^\cap_y \in \boldsymbol{T}^\cap$ such that $t^\cap_x \subseteq t^\cap_y$. Then $\forall t_x \in t^\cap_x, t_x \in t^\cap_y$.
Thus $\forall t_x \in t^\cap_x, \exists t_y (= t_x) \in t^\cap_y : t_y \lesssim t_x$, so by proposition 4.5, $t^\cap_y \overset{\cap}{\lesssim} t^\cap_x$. □

The set of comparisons $\boldsymbol{C}^\cap$ is constructed such that the top (the less specific CUTs) and the bottom (the most specific CUTs) of the pre-ordered set $(\boldsymbol{T}^\cap, \overset{\cap}{\lesssim})$ is flattened. The following proposition underlines the important maximal and minimal elements of $(\boldsymbol{T}^\cap, \overset{\cap}{\lesssim})$.

Proposition 4.7. $\top^\cap$ *and* $\varnothing$ *are maximal elements in* $\mathbf{T}^\cap$, *and* $\mathbf{T}$, $\bot^\cap$, $\bot^\sim$, *and for all* $s \in \mathbf{S}_\mathcal{T}, \{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\}$ *are minimal elements in* $(\mathbf{T}^\cap, \overset{\cap}{\lesssim})$.

Proof. Let $t^\cap \in \boldsymbol{T}^\cap$.
$\varnothing$: $\varnothing \subseteq t^\cap$, so using proposition 4.6: $t^\cap \overset{\cap}{\lesssim} \varnothing$.
$\top^\cap$: Let $t^\cap \neq \varnothing$. We know that for all $t \in \boldsymbol{T}$, $t \lesssim \top$, so for $\top$ in $\top^\cap$, $\exists t \in t^\cap : t \lesssim \top$.
So using proposition 4.5, $t^\cap \overset{\cap}{\lesssim} \top^\cap$.
As we moreover know that $\varnothing \overset{\cap}{\lesssim} \top^\cap$, then for all $t^\cap \in \boldsymbol{T}^\cap$, $t^\cap \overset{\cap}{\lesssim} \top^\cap$.
$\boldsymbol{T}$: $t^\cap \subseteq \boldsymbol{T}$, so using proposition 4.6: $\boldsymbol{T} \overset{\cap}{\lesssim} t^\cap$.
$\bot^\cap$: Let $t^\cap \neq \varnothing$. We know that for all $t \in \boldsymbol{T}$, $\bot \lesssim t$, so for all $t \in t^\cap, \bot \lesssim t$.
So using proposition 4.5, $\bot^\cap \overset{\cap}{\lesssim} t^\cap$.
As we moreover know that $\varnothing$ is a maximal element, $\bot^\cap \overset{\cap}{\lesssim} \varnothing$, then for all $t^\cap \in \boldsymbol{T}^\cap$, $\bot^\cap \overset{\cap}{\lesssim} t^\cap$.
$\bot^\sim$: We know that $\bot \in \bot^\sim$, so item above also applies for $\bot^\sim$, i.e.,
for $t^\cap \neq \varnothing$, $\forall t \in t^\cap, \exists t' (= \bot) \in \bot^\sim : t' \lesssim t$, and $\bot^\sim \overset{\cap}{\lesssim} \varnothing$
So for all $t^\cap \in \boldsymbol{T}^\cap$, $\bot^\sim \overset{\cap}{\lesssim} t^\cap$.
$\{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\}$: Let $s \in \boldsymbol{S}_\mathcal{T}$. We know that $(\bot^\cap, \{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\}) \in \boldsymbol{C}^\cap_\mathbf{\Gamma}$, so $\{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\} \overset{\cap}{\lesssim} \bot^\cap$, and as $\bot^\cap$ is a minimal element, so is $\{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\}$. □

We introduce the natural equivalence relation $\overset{\cap}{\simeq}$ defined by $t^\cap_x \overset{\cap}{\simeq} t^\cap_y \Leftrightarrow t^\cap_x \overset{\cap}{\lesssim} t^\cap_y$ and $t^\cap_y \overset{\cap}{\lesssim} t^\cap_x$. Elements in the flattened bottom of the pre-ordered set $(\boldsymbol{T}^\cap, \overset{\cap}{\lesssim})$ are not only the most specific CUTs, but they all are considered absurd, and may not have instances.

Definition 4.10 (Absurd CUT set)**.** The set of absurd CUTs is denoted by $\bot^\sqcap$ and is the set: $\bot^\sqcap \overset{\text{def}}{=} \{t^\cap \in \boldsymbol{T}^\cap \mid t^\cap \overset{\cap}{\simeq} \bot^\cap\}$.

Naturally, the prime absurd CUT, $\bot^\cap$, is absurd.

The property of being absurd is hereditary: if a CUT $t^\cap$ is absurd, all conjunctive types lesser than $t^\cap$ are also absurd. For instance if $\{def, indef\}$ is asserted to be absurd and USA $\lesssim def$, then $\{\text{USA}, indef\}$ is absurd.

There are two desired behaviours that are not automatic for PUT.

First, if a PUT $t \in \boldsymbol{T}$ has a ASlot s both obligatory and prohibited, then it shall be absurd. This is not automatic considering the PUTs t. But consider now the primitive CUT $\{t\} \in \boldsymbol{T}^\cap$.

As $t \lesssim \boldsymbol{\gamma_1}(s)$ and $t \lesssim \boldsymbol{\gamma_0}(s)$, then one can prove that $\{t\} \overset{\cap}{\lesssim} \{\boldsymbol{\gamma_1}(s), \boldsymbol{\gamma_0}(s)\}$, and thus $\{t\} \overset{\cap}{\lesssim} \bot^{\cap}$, so $\{t\}$ is absurd.

Second, if a PUT has an absurd signature, then it is absurd. Consider for instance that one wants to define $^{/}$rain-hail$^{\backslash}$ as being a specialization of both $^{/}$rain$^{\backslash}$ and $^{/}$hail$^{\backslash}$. From definition 3.12, $^{/}$rain-hail$^{\backslash}$ has a ASlot *obj*, whose signature is the union of those of $^{/}$rain$^{\backslash}$ and $^{/}$hail$^{\backslash}$:

$$\begin{aligned}\boldsymbol{\varsigma}_{/\text{rain-hail}\backslash}(obj) &= \boldsymbol{\varsigma}_{/\text{rain}\backslash}(obj) \cup \boldsymbol{\varsigma}_{/\text{hail}\backslash}(obj) \\ &= \{/\text{water}\backslash, /\text{liquid}\backslash\} \cup \{/\text{water}\backslash, /\text{solid}\backslash\} \\ &= \{/\text{water}\backslash, /\text{liquid}\backslash, /\text{solid}\backslash\}\end{aligned}$$

If one asserted that $\{/\text{liquid}\backslash, /\text{solid}\backslash\}$ is absurd and as being absurd is hereditary, the signature of PUT $^{/}$rain-hail$^{\backslash}$ is absurd. Yet, PUT is not automatically absurd. Equation 17 in the definition of the pre-order $\overset{\cap}{\lesssim}$ enables PUT $\{/\text{rain-hail}\backslash\}$ to be absurd as it should be.

Finally, if one asserts that $\top^{\cap} \in \bot_A^{\sqcap}$, then the whole hierarchy collapses and $\bot^{\sqcap} = \boldsymbol{T}^{\cap}$. Same goes if $t^{\cap} \in \boldsymbol{T}^{\cap}$ such that $\top^{\cap} \overset{\cap}{\lesssim} t^{\cap}$, and $s \in \boldsymbol{\alpha}^{\cap}(t^{\cap})$ such that $\boldsymbol{\varsigma}^{\cap}_{t^{\cap}}(s) \overset{\cap}{\lesssim} \bot^{\cap}$. Then from proposition 4.5, $t^{\cap} \overset{\cap}{\lesssim} \bot^{\cap}$, and by the pre-order transitivity, $\bot^{\sqcap} = \boldsymbol{T}^{\cap}$. Such situations must absolutely be avoided.

4.4 Hierarchy of Unit Types

We are now ready to introduce the unit types hierarchy, core of the UGs mathematical framework.

Definition 4.11 (CUT hierarchy)**.** A *hierarchy of unit types* $\mathcal{T} = (T_D, \boldsymbol{S}_\mathcal{T}, \boldsymbol{\Gamma}, \boldsymbol{\gamma}, \boldsymbol{\Gamma_1}, \boldsymbol{\gamma_1}, \boldsymbol{\Gamma_0}, \boldsymbol{\gamma_0}, C_A, \bot^{\sqcap}_A, \{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}})$, is composed of:

- T_D a set of declared PUTs;
- $\boldsymbol{S}_\mathcal{T}$ the set of ASymbols;
- $\boldsymbol{\Gamma}$ a set of radices PUTs;
- $\boldsymbol{\gamma}$ the bijection from $\boldsymbol{S}_\mathcal{T}$ to $\boldsymbol{\Gamma}$;
- $\boldsymbol{\Gamma_1}$ a set of obligant PUTs;
- $\boldsymbol{\gamma_1}$ the bijection from $\boldsymbol{S}_\mathcal{T}$ to $\boldsymbol{\Gamma_1}$;
- $\boldsymbol{\Gamma_0}$ a set of prohibent PUTs;
- $\boldsymbol{\gamma_0}$ the bijection from $\boldsymbol{S}_\mathcal{T}$ to $\boldsymbol{\Gamma_0}$;
- C_A a set of asserted PUTs comparisons;
- $\bot^{\sqcap}_A$ a set of asserted absurd CUTs;
- $\{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}}$ the set of signatures of PUTs;

The CUTs hierarchy is the minimal set of mathematical objects that is necessary to form the consistent core of the UG formalism: a set of CUTs with actantial structure.

5 Characterizing CUTs

This section introduces remarkable aspects of the CUTs.

First, for two CUTs $t_x^\cap, t_y^\cap \in \boldsymbol{T}^\cap$ and according to the construction of the pre-order over CUTs, we introduce in §5.1 the necessary condition to have $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$.

Second, now that the pre-order $\overset{\cap}{\lesssim}$ is introduced, we will see in §5.2 that most properties of PUTs slots and signatures are valid for CUTs slots and signatures, except for some degenerate cases.

Next, §5.3 introduces a natural equivalence relation over the set of CUTs, and the partially ordered set of equivalent classes of CUTs.

Finally, §5.4 and 5.5 are devoted to the introduction of remarkable subsets of CUTs that will be important later on for the UGs mathematical framework.

5.1 Necessary Conditions to Compare Two CUTs

The pre-order $\overset{\cap}{\lesssim}$ over CUTs is slightly more complex than the simple natural extension of each pre-order over a set to a pre-order over its powerset. The necessary condition to have two CUTs $t_x^\cap, t_y^\cap \in \boldsymbol{T}^\cap$ comparable may be stated as follows:

If $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$, then at least one of the following is true:

- $t_x^\cap$ is absurd;
- $t_x^\cap$ is the empty set (and thus $t_y^\cap$ is a maximal element of $\boldsymbol{T}^\cap$);
- $t_x^\cap$ is naturally more specific than $t_y^\cap$, i.e., according to the natural extension of each pre-order over a set to a pre-order over its powerset.

Or more formally:

Proposition 5.1. *Let $t_x^\cap, t_y^\cap \in \mathbf{T}^\cap$. if $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$, then at least one of the following is true:*

- $t_x^\cap \overset{\cap}{\simeq} \bot^\cap$;
- $t_x^\cap = \varnothing$ *(and thus* $\top^\cap \overset{\cap}{\simeq} t_y^\cap$*);*
- $\forall t_y \in t_y^\cap, \exists t_x \in t_x^\cap : t_x \lesssim t_y$ *(i.e.,* $(t_y^\cap, t_x^\cap) \in \mathbf{C}_{\overset{\cap}{\lesssim}}^\cap$*).*

Proof. Let $t_x^\cap, t_y^\cap \in \boldsymbol{T}^\cap$.
As $\bot^\cap$ is a minimal) element of $\boldsymbol{T}^\cap$, $t_x^\cap \overset{\cap}{\lesssim} \bot^\cap$ implies $t_x^\cap \overset{\cap}{\simeq} \bot^\cap$.
So it is sufficient to prove that one of the following is true:
i) $t_x^\cap \overset{\cap}{\lesssim} \bot^\cap$, ii) $t_x^\cap = \varnothing$ (and thus $\top^\cap \overset{\cap}{\simeq} t_y^\cap$), or iii) $(t_y^\cap, t_x^\cap) \in \boldsymbol{C}_{\overset{\cap}{\lesssim}}^\cap$.

$\boldsymbol{C}^\cap$ is the limit the sequence $(\boldsymbol{C}_n^\cap)$, so let $n \in \mathbb{N}$ be such that $(t_y^\cap, t_x^\cap) \in \boldsymbol{C}_n^\cap$ and $(t_y^\cap, t_x^\cap) \notin \boldsymbol{C}_{n-1}^\cap$.
We assume that $t_x^\cap \overset{\cap}{\not\lesssim} \bot^\cap$, $(t_y^\cap, t_x^\cap) \notin \boldsymbol{C}_{\overset{\cap}{\lesssim}}^\cap$, and that either $\top^\cap \overset{\cap}{\not\lesssim} t_y^\cap$ or $t_x^\cap \neq \varnothing$.

1) If $(t_y^\cap, t_x^\cap) \in \boldsymbol{C}_{\boldsymbol{\varsigma}\,n}^\cap$, then $t_x^\cap \overset{\cap}{\lesssim} \bot^\cap$.
This is impossible, so $(t_y^\cap, t_x^\cap) \in \boldsymbol{C}_{+n}^\cap$ and there exists $t^\cap \in \boldsymbol{T}^\cap$ such that $(t_y^\cap, t^\cap) \in \boldsymbol{C}_{n-1}^\cap$ and $(t^\cap, t_x^\cap) \in \boldsymbol{C}_{n-1}^\cap$.
As $\boldsymbol{C}_{n-1}^\cap \subseteq \boldsymbol{C}^\cap$, $t_x^\cap \overset{\cap}{\lesssim} t^\cap$.
If $t^\cap \overset{\cap}{\lesssim} \bot^\cap$, then $t_x^\cap \overset{\cap}{\lesssim} \bot^\cap$ which is impossible.
So we know that $t^\cap \overset{\cap}{\not\lesssim} \bot^\cap$.

2) From proposition 4.3, we know that $\boldsymbol{C}^{\cap}_{\overset{\cap}{\lesssim}}$ is transitive. So either 2a) $(t^{\cap}, t^{\cap}_x) \notin \boldsymbol{C}^{\cap}_{\overset{\cap}{\lesssim}}$, or 2b) $(t^{\cap}_y, t^{\cap}) \notin \boldsymbol{C}^{\cap}_{\overset{\cap}{\lesssim}}$.

2a) In this case, consider $m \leq n-1$ such that $(t^{\cap}, t^{\cap}_x) \in \boldsymbol{C}^{\cap}_m$ and $(t^{\cap}, t^{\cap}_x) \notin \boldsymbol{C}^{\cap}_{m-1}$.
We still have $t^{\cap}_x \overset{\cap}{\not\lesssim} \bot^{\cap}$ and we have now $(t^{\cap}, t^{\cap}_x) \notin \boldsymbol{C}^{\cap}_{\overset{\cap}{\lesssim}}$.

2b) In this case, consider $m \leq n-1$ such that $(t^{\cap}_y, t^{\cap}) \in \boldsymbol{C}^{\cap}_m$ and $(t^{\cap}_y, t^{\cap}) \notin \boldsymbol{C}^{\cap}_{m-1}$.
We just showed in 1) that $t^{\cap} \overset{\cap}{\not\lesssim} \bot^{\cap}$ and we have now $(t^{\cap}_y, t^{\cap}) \notin \boldsymbol{C}^{\cap}_{\overset{\cap}{\lesssim}}$.

3) In both cases 2a) and 2b), the possibility of our hypothesis depend on the possibility of our hypothesis with different CUTs, and with a strictly lower natural number $m < n$.
Thus for each exploratory branch, at some point we will reach the case where (for instance for $t^{\cap}_a$ and $t^{\cap}_b$): $(t^{\cap}_b, t^{\cap}_a) \in \boldsymbol{C}^{\cap}_0$, $t^{\cap}_a \overset{\cap}{\not\lesssim} \bot^{\cap}$ so $(\bot^{\cap}, t^{\cap}_a) \notin \boldsymbol{C}^{\cap}_{\bot}$, $(\bot^{\cap}, t^{\cap}_a) \notin \boldsymbol{C}^{\cap}_{\mathbf{T}}$, and $(t^{\cap}_b, t^{\cap}_a) \notin \boldsymbol{C}^{\cap}_{\overset{\cap}{\lesssim}}$. So $(t^{\cap}_b, t^{\cap}_a) \in \boldsymbol{C}^{\cap}_{\top}$.
Neither $\top^{\cap} \overset{\cap}{\not\lesssim} t^{\cap}_b$ nor $t^{\cap}_b \neq \varnothing$ is compatible with this last statement, so we highlighted a contradiction.

As a conclusion, if $t^{\cap}_x \overset{\cap}{\lesssim} t^{\cap}_y$, it is impossible that all of the following is false :
i) $t^{\cap}_x \overset{\cap}{\lesssim} \bot^{\cap}$, ii) $t^{\cap}_x = \varnothing$, iii) $(t^{\cap}_y, t^{\cap}_x) \in \boldsymbol{C}^{\cap}_{\overset{\cap}{\lesssim}}$. □

5.2 Properties of CUT Slots and Signatures

Now that the pre-order $\overset{\cap}{\lesssim}$ is introduced, we will see how the properties of PUTs slots and signatures are valid for CUTs slots and signatures except for specific degenerate cases: the void CUT and absurd CUTs.

ASlots, OblASlots, and ProASlots First, remember proposition 3.3 and the fact that the ASlot (resp1. OblASlot, resp2. ProASlots) *obj* is inherited by every PUT more specific than $\boldsymbol{\gamma}(obj)$ (resp1. $\boldsymbol{\gamma_1}(s)$, resp2. $\boldsymbol{\gamma_0}(s)$), and only those PUTs more specific than $\boldsymbol{\gamma}(obj)$ (resp1. $\boldsymbol{\gamma_1}(s)$, resp2. $\boldsymbol{\gamma_0}(s)$) have a ASlot (resp1. OblASlot, resp2. ProASlots) with symbol *obj*. From the definition of ASlots (resp1. OblASlot, resp2. ProASlots) of CUTs, this property is also valid for CUTs:

Proposition 5.2. *Let $\downarrow^{\cap} t$ be the smallest lower set of $\mathbf{T}^{\cap}$ that contains $\{t\}$:*

$$\downarrow^{\cap} t \overset{def}{=} \{t^{\cap} \in \mathbf{T}^{\cap} \mid t^{\cap} \overset{\cap}{\lesssim} \{t\}\} \tag{19}$$

For any $s \in \mathbf{S}_{\mathcal{T}}$, the following holds:

$$\{t^{\cap} \in \mathbf{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing \quad = \downarrow^{\cap} \boldsymbol{\gamma}(s) \setminus \varnothing \tag{20}$$

$$\{t^{\cap} \in \mathbf{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}_{\boldsymbol{1}}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing \quad = \downarrow^{\cap} \boldsymbol{\gamma_1}(s) \setminus \varnothing \tag{21}$$

$$\{t^{\cap} \in \mathbf{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}_{\boldsymbol{0}}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing \quad = \downarrow^{\cap} \boldsymbol{\gamma_0}(s) \setminus \varnothing \tag{22}$$

Proof. Let $s \in \boldsymbol{S}_{\mathcal{T}}$.
$\subseteq_1$: Let $t^{\cap} \in \{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}(t^{\cap})\}$.
By definition 4.3, $\exists t \in t^{\cap}, s \in \boldsymbol{\alpha}(t)$. So $t^{\cap} \neq \varnothing$.
On the other hand, from definition 3.8, $t \lesssim \boldsymbol{\gamma}(s)$.
By proposition 4.5, $t^{\cap} \overset{\cap}{\lesssim} \{\boldsymbol{\gamma}(s)\}$ so $t^{\cap} \in \downarrow^{\cap} \boldsymbol{\gamma}(s)$.
So $t^{\cap} \in \downarrow^{\cap} \boldsymbol{\gamma}(s) \setminus \varnothing$, and $\{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}(t^{\cap})\} \subseteq \downarrow^{\cap} \boldsymbol{\gamma}(s) \setminus \varnothing$.

$\subseteq_2$: Let $t^\cap \in \bot^\sqcap \setminus \varnothing$. $t^\cap \overset{\cap}{\lesssim} \bot^\cap$ which is a minimum element in $\boldsymbol{T}^\cap$.
So $t^\cap \overset{\cap}{\lesssim} \{\boldsymbol{\gamma}(s)\}$ and $t^\cap \in \downarrow^\cap \boldsymbol{\gamma}(s)$ and $t^\cap \in \downarrow^\cap \boldsymbol{\gamma}(s) \setminus \varnothing$.
So $\bot^\sqcap \setminus \varnothing \subseteq \downarrow^\cap \boldsymbol{\gamma}(s) \setminus \varnothing$.
$\supseteq$: Let $t^\cap \in \downarrow^\cap \boldsymbol{\gamma}(s) \setminus \varnothing$. $t^\cap \overset{\cap}{\lesssim} \{\boldsymbol{\gamma}(s)\}$.
We use proposition 5.1, so at least one of the following is true:

i) $t^\cap \overset{\cap}{\lesssim} \bot^\cap$: then $t^\cap \in \bot^\sqcap$. And as $t^\cap \neq \varnothing$, $t^\cap \in \bot^\sqcap \setminus \varnothing$.

ii) $t^\cap = \varnothing$: impossible case.

iii) $\exists t \in t^\cap : t \lesssim \boldsymbol{\gamma}(s)$: then using definition 3.8, $\exists t \in t^\cap : s \in \boldsymbol{\alpha}(t)$, and by definition 4.3, $s \in \boldsymbol{\alpha}^\cap(t^\cap)$. So $t^\cap \in \{t^\cap \in \boldsymbol{T}^\cap \mid s \in \boldsymbol{\alpha}^\cap(t^\cap)\}$.

As the definitions of ASlots, OblASlots and ProASlots are parallel, the proofs for OblASlots and ProASlots are exactly the same, except for symbols and references to definitions. □

One may have noticed that two kinds of CUTs are not considered here: the CUT $\varnothing$, and any absurd CUT $t^\cap \in \bot^\sqcap$:

- the CUT $\varnothing$: a problem would arise for instance with $s \in \boldsymbol{S_\mathcal{T}}$ such that $\boldsymbol{\gamma}(s) = \top$. By construction of the pre-order $\overset{\cap}{\lesssim}$, $\varnothing \overset{\cap}{\lesssim} \top^\cap$, yet $\varnothing$ has no ASlot ($\boldsymbol{\alpha}^\cap(\varnothing) = \bigcup_{t\in\varnothing} \boldsymbol{\alpha}(t) = \varnothing$). So $s \notin \boldsymbol{\alpha}^\cap(\varnothing)$.

- any absurd CUT $t^\cap \in \bot^\sqcap$: problems arise for any $s \in \boldsymbol{S_\mathcal{T}}$ such that $s \notin \boldsymbol{\alpha}^\cap(t^\cap)$. By construction of the pre-order $\overset{\cap}{\lesssim}$, $t^\cap \overset{\cap}{\lesssim} \bot^\cap$, yet, from propositions 3.6 and 4.2, $\boldsymbol{\alpha}^\cap(\bot^\cap) = \boldsymbol{S_\mathcal{T}}$. So some ASlots are not inherited among absurd CUTs.

In fact, these two degenerate cases are not considered in all of the following properties of this section.

As CUTs get more and more specific (i.e., as we go down the hierarchy of CUTs), the set of ASlots (resp1. OblASlots, resp2. ProASlots) may only increase. ASlots (resp1. OblASlots, resp2. ProASlots) are inherited also for CUTs, as long as they are not the empty CUT, or absurd types.

Proposition 5.3. *ASlots, OblASlots and ProASlots are inherited as long as CUTs are not absurd and are not $\varnothing$, i.e., $\forall t_x^\cap \in \mathbf{T}^\cap \setminus \varnothing - \bot^\sqcap$, and $t_y^\cap \in \mathbf{T}^\cap$,*

$$t_x^\cap \overset{\cap}{\lesssim} t_y^\cap \Rightarrow \boldsymbol{\alpha}^\cap(t_y^\cap) \subseteq \boldsymbol{\alpha}^\cap(t_x^\cap) \tag{23}$$

$$t_x^\cap \overset{\cap}{\lesssim} t_y^\cap \Rightarrow \boldsymbol{\alpha_1^\cap}(t_y^\cap) \subseteq \boldsymbol{\alpha_1^\cap}(t_x^\cap) \tag{24}$$

$$t_x^\cap \overset{\cap}{\lesssim} t_y^\cap \Rightarrow \boldsymbol{\alpha_0^\cap}(t_y^\cap) \subseteq \boldsymbol{\alpha_0^\cap}(t_x^\cap) \tag{25}$$

Proof. Let $t_x^\cap \in \boldsymbol{T}^\cap \setminus \varnothing - \bot^\sqcap$ and $t_y^\cap \in \boldsymbol{T}^\cap$ such that $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$.
Let $s \in \boldsymbol{\alpha}^\cap(t_y^\cap)$. By definition 4.3, $\exists t \in t_y^\cap, s \in \boldsymbol{\alpha}(t)$.
We use proposition 5.1, but as $t_x^\cap \notin \bot^\sqcap$ and $t_x^\cap \neq \varnothing$, $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$ simply implies: $\forall t_y \in t_y^\cap, \exists t_x \in t_x^\cap : t_x \lesssim t$. So $\exists t_x \in t_x^\cap : t_x \lesssim t_y$.
Using proposition 3.4, $s \in \boldsymbol{\alpha}(t_x)$, and using definition 4.3, $s \in \boldsymbol{\alpha}^\cap(t_x^\cap)$.
As the definitions of ASlots, OblASlots and ProASlots are parallel, the proofs for OblASlots and ProASlots are exactly the same, except for symbols and references to definitions. □

OptASlots Furthermore, due to proposition 5.2, we know that apart from absurd CUTs and $\varnothing$, any CUT with OptASlot *obj* is in lower set $\downarrow^{\cap}\boldsymbol{\gamma}(obj)$ and is not in lower sets $\downarrow^{\cap}\boldsymbol{\gamma_1}(obj)$ and $\downarrow^{\cap}\boldsymbol{\gamma_0}(obj)$. So the following proposition holds:

Proposition 5.4. *For any* $s \in \mathbf{S}_{\mathcal{T}}$,

$$\begin{aligned}
&\{t^{\cap} \in \mathbf{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}_{\boldsymbol{?}}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing \\
&= (\downarrow^{\cap}\boldsymbol{\gamma}(s) \setminus \varnothing - \downarrow^{\cap}\boldsymbol{\gamma_1}(s) \setminus \varnothing - \downarrow^{\cap}\boldsymbol{\gamma_0}(s) \setminus \varnothing) \cup \bot^{\sqcap} \setminus \varnothing \\
&= \left\{t^{\cap} \in \mathbf{T}^{\cap} \mid t^{\cap} \overset{\cap}{\lesssim} \{\boldsymbol{\gamma}(s)\} \text{ and } t^{\cap} \overset{\cap}{\not\lesssim} \{\boldsymbol{\gamma_1}(s)\} \text{ and } t^{\cap} \overset{\cap}{\not\lesssim} \{\boldsymbol{\gamma_0}(s)\}\right\} \cup \bot^{\sqcap} \setminus \varnothing
\end{aligned}$$

Proof. We use propositions 3.7 and 5.2:

$$\begin{aligned}
&\{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}_{\boldsymbol{?}}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing \\
&= \{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}(t^{\cap}) - \boldsymbol{\alpha}^{\cap}_{\boldsymbol{1}}(t^{\cap}) - \boldsymbol{\alpha}^{\cap}_{\boldsymbol{0}}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing \\
&= \big(\{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}(t^{\cap})\} - \{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}_{\boldsymbol{1}}(t^{\cap})\} - \{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}_{\boldsymbol{0}}(t^{\cap})\}\big) \cup \bot^{\sqcap} \setminus \varnothing \\
&= \big(\big(\{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing\big) - \big(\{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}_{\boldsymbol{1}}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing\big) - \big(\{t^{\cap} \in \boldsymbol{T}^{\cap} \mid s \in \boldsymbol{\alpha}^{\cap}_{\boldsymbol{0}}(t^{\cap})\} \cup \bot^{\sqcap} \setminus \varnothing \\
&= (\downarrow^{\cap}\boldsymbol{\gamma}(s) \setminus \varnothing - \downarrow^{\cap}\boldsymbol{\gamma_1}(s) \setminus \varnothing - \downarrow^{\cap}\boldsymbol{\gamma_0}(s) \setminus \varnothing) \cup \bot^{\sqcap} \setminus \varnothing
\end{aligned}$$

□

Signatures Just like for PUTs, as CUTs get more and more specific, the signature of a given common ASlot may only become more and more specific:

Proposition 5.5. *For all* $t^{\cap}_x \in \mathbf{T}^{\cap} \setminus \varnothing - \bot^{\sqcap}$, $t^{\cap}_y \in \mathbf{T}^{\cap}$ *and* $s \in \mathbf{S}_{\mathcal{T}}$ *such that* $s \in \boldsymbol{\alpha}^{\cap}(t^{\cap}_y)$,

$$t^{\cap}_x \overset{\cap}{\lesssim} t^{\cap}_y \Rightarrow \boldsymbol{\varsigma}^{\cap}_{t^{\cap}_x}(s) \overset{\cap}{\lesssim} \boldsymbol{\varsigma}^{\cap}_{t^{\cap}_y}(s) \tag{26}$$

Proof. Let $t^{\cap}_x \in \boldsymbol{T}^{\cap} \setminus \varnothing - \bot^{\sqcap}$ and $t^{\cap}_y \in \boldsymbol{T}^{\cap}$ and $s \in \boldsymbol{S}_{\mathcal{T}}$ such that $t^{\cap}_x \overset{\cap}{\lesssim} t^{\cap}_y$ and $s \in \boldsymbol{\alpha}^{\cap}(t^{\cap}_y)$.
1) Let $t_a \in \boldsymbol{\varsigma}^{\cap}_{t^{\cap}_y}(s)$. Using definition 4.7, $\exists t_b \in t^{\cap}_y : s \in \boldsymbol{\alpha}(t_b)$ and $t_a \in \boldsymbol{\varsigma}_{t_b}(s)$.
2) We use $t^{\cap}_x \overset{\cap}{\lesssim} t^{\cap}_y$ and proposition 5.1.
As $t^{\cap}_x \notin \bot^{\sqcap}$ and $t^{\cap}_x \neq \varnothing$, we know that $\forall t_y \in t^{\cap}_y, \exists t_x \in t^{\cap}_x : t_x \lesssim t_y$.
We restrict those t_y to be such that $s \in \boldsymbol{\alpha}(t_y)$.
We use definitions 3.12 and proposition 4.5: $\forall t_y \in t^{\cap}_y$ such that $s \in \boldsymbol{\alpha}(t_y)$, there exists $t_x \in t^{\cap}_x$ such that $\forall t' \in \boldsymbol{\varsigma}_{t_y}(s), \exists t \in \boldsymbol{\varsigma}_{t_x}(s) : t \lesssim t'$
3) This is rewritten as: $\forall t_y \in t^{\cap}_y$ such that $s \in \boldsymbol{\alpha}(t_y), \forall t' \in \boldsymbol{\varsigma}_{t_y}(s), \exists t_x, t$ such that $t_x \in t^{\cap}_x, t \in \boldsymbol{\varsigma}_{t_x}(s)$ and $t \lesssim t'$
4) combining 1) and 3) with $t_y = t_b$, and $t' = t_a$, one obtains:
$\exists t_b \in t^{\cap}_y$ such that $s \in \boldsymbol{\alpha}(t_b)$ and $t_a \in \boldsymbol{\varsigma}_{t_b}(s), \exists t_x, t$ such that $t_x \in t^{\cap}_x, t \in \boldsymbol{\varsigma}_{t_x}(s)$ and $t \lesssim t_a$
5) So there exists t and t_x such that $t_x \in t^{\cap}_x$ and $t \in \boldsymbol{\varsigma}_{t_x}(s)$ and $t \lesssim t_a$,
6) using definition 4.7, $t \in \boldsymbol{\varsigma}^{\cap}_{t^{\cap}_x}(s)$. So $\forall t_a \in \boldsymbol{\varsigma}^{\cap}_{t^{\cap}_y}(s), \exists t \in \boldsymbol{\varsigma}^{\cap}_{t^{\cap}_x}(s) : t \lesssim t_a$, and using proposition 4.5, $\boldsymbol{\varsigma}^{\cap}_{t^{\cap}_x}(s) \overset{\cap}{\lesssim} \boldsymbol{\varsigma}^{\cap}_{t^{\cap}_y}(s)$. □

5.3 CUT Equivalence Class Sets

Let $\overset{\cap}{\simeq}$ be the natural *equivalence* relation defined by $t_x^\cap \overset{\cap}{\simeq} t_y^\cap \Leftrightarrow \left(t_x^\cap \overset{\cap}{\lesssim} t_y^\cap \wedge t_y^\cap \overset{\cap}{\lesssim} t_x^\cap\right)$. The set of equivalence classes defines a partition of $\boldsymbol{T}^\cap$. Let $t^\cap \in \boldsymbol{T}^\cap$, we denote $[t^\cap] \overset{\text{def}}{=} \{t_x^\cap \in \boldsymbol{T}^\cap \mid t_x^\cap \overset{\cap}{\simeq} t^\cap\}$ the equivalence class to which $t^\cap$ belongs. We will usually use the notation $t^\sqcap$ for an unknown equivalence class.

Definition 5.1 (CUTs equivalence class set)**.** The *CUTs equivalence class set* $\boldsymbol{T}^\sqcap$ is the quotient set of $\boldsymbol{T}^\cap$ by $\overset{\cap}{\simeq}$, i.e., $\boldsymbol{T}^\sqcap \overset{\text{def}}{=} \boldsymbol{T}^\cap/\overset{\cap}{\simeq} = \{[t^\cap] \mid t^\cap \in \boldsymbol{T}^\cap\}$. We define a partial order $\overset{\sqcap}{\leq}$ over $\boldsymbol{T}^\sqcap$ with $[t_x^\cap] \overset{\sqcap}{\leq} [t_y^\cap]$ if and only if $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$.

We define $\top^\sqcap \overset{\text{def}}{=} [\top^\cap]$ the top (less specific) CUTs equivalence class set, and $\bot^\sqcap \overset{\text{def}}{=} [\bot^\cap]$ the absurd (most specific) CUTs equivalence class set.

Proposition 5.6. *$\top^\sqcap$ and $\bot^\sqcap$ are respectively the greatest element and the least element of $\mathbf{T}^\sqcap$.*

Proof. $\top^\sqcap$: From proposition 4.7, $\top^\cap$ is a maximal element for $\overset{\cap}{\lesssim}$.
So $\forall t^\cap \in \boldsymbol{T}^\cap, t^\cap \overset{\cap}{\lesssim} \top^\cap$. So $\forall t^\sqcap \in \boldsymbol{T}^\sqcap, t^\sqcap \overset{\sqcap}{\leq} \top^\sqcap$, and $\top^\sqcap$ is the greatest element of $\boldsymbol{T}^\sqcap$.
$\bot^\sqcap$: From proposition 4.7, $\bot^\cap$ is a minimal element for $\overset{\cap}{\lesssim}$. So $\forall t^\cap \in \boldsymbol{T}^\cap, \bot^\cap \overset{\cap}{\lesssim} t^\cap$.
So $\forall t^\sqcap \in \boldsymbol{T}^\sqcap, \bot^\sqcap \overset{\sqcap}{\leq} t^\sqcap$, so $\bot^\sqcap$ is the least element of $\boldsymbol{T}^\sqcap$. □

A set of equivalence class CUTs is closed under the union operation:

Proposition 5.7. *Let $t^\sqcap \in \mathbf{T}^\sqcap$, and $t_x^\cap, t_y^\cap \in t^\sqcap$. Then $t_x^\cap \cup t_y^\cap \in t^\sqcap$.*

Proof. Let $t^\sqcap \in \boldsymbol{T}^\sqcap$, and $t_x^\cap, t_y^\cap \in t^\sqcap$. We will prove that $t_x^\cap \cup t_y^\cap \overset{\cap}{\simeq} t_x^\cap$.
$\overset{\cap}{\lesssim}$: We know from proposition 4.6 that $t_x^\cap \subseteq t_x^\cap \cup t_y^\cap$ implies $t_x^\cap \cup t_y^\cap \overset{\cap}{\lesssim} t_x^\cap$.
$\overset{\cap}{\gtrsim}$: We know that $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$. From proposition 4.5, at least one of the followings holds:
i) $t_x^\cap \overset{\cap}{\lesssim} \bot^\cap$: then $t_x^\cap \cup t_y^\cap \overset{\cap}{\simeq} t_x^\cap \overset{\cap}{\simeq} \bot^\cap$;
ii) $t_x^\cap = \varnothing$: then $t_x^\cap \cup t_y^\cap = t_y^\cap \overset{\cap}{\simeq} t_x^\cap$;
iii) $\forall t_y \in t_y^\cap, \exists t_x \in t_x^\cap : t_x \lesssim t_y$: So $\forall t \in t_x^\cap \cup t_y^\cap, \exists t_x \in t_x^\cap : t_x \lesssim t$, and $t_x^\cap \overset{\cap}{\lesssim} t_x^\cap \cup t_y^\cap$. □

The following lemma is useful to prove upcoming propositions. It states that adding a PUT t to a CUT $t^\cap$ does not make it change its equivalence class, provided that t is greater than at least one PUT in $t^\cap$:

Lemma 5.8. *Let $t^\cap \in \mathbf{T}^\cap$ and $t \in \mathbf{T}$ such that $\exists t' \in t^\cap, t' \lesssim t$. Then $t^\cap \cup \{t\} \overset{\cap}{\simeq} t^\cap$.*

Proof. Let $t^\cap \in \boldsymbol{T}^\cap$ and $t \in \boldsymbol{T}$ such that $\forall t' \in t^\cap, t' \lesssim t$.
$\overset{\cap}{\lesssim}$: $t_y^\cap \subseteq t_y^\cap \cup \{t\}$ so $t^\cap \cup \{t\} \overset{\cap}{\lesssim} t^\cap$;
$\overset{\cap}{\gtrsim}$: $t^\cap \neq \varnothing$. Let $t' \in t^\cap \cup \{t\}$.
If $t' \in t^\cap$, then there exists $t''(= t') \in t^\cap : t'' \lesssim t'$.
If $t' = t$, then there exists $t'' \in t^\cap t'' \lesssim t'$.
So $\forall t' \in t^\cap \cup \{t\}, \exists t'' \in t^\cap : t'' \lesssim t'$, and $t^\cap \cup \{t\} \overset{\cap}{\gtrsim} t^\cap$. □

5.4 Maximal CUTs

CUTs enable a first inference mechanism, which is type inference. If a CUT $t_x^\cap$ is lower than $t_y^\cap$, then any instance of $t_x^\cap$ is also an instance of every PUT in $t_y^\cap$. In this section we consider a remarkable subset of CUTs that we call the set of maximal CUTs.

Let us first introduce a maximization operator $\Uparrow$ that associates to any CUT $t^\cap$ the union of all CUTs in $[t^\cap]$:

Definition 5.2 (Maximization operator)**.** The maximization operator $\Uparrow$ defined on $\boldsymbol{T}^\cap$ associates to every CUT $t^\cap$ the CUT $\Uparrow t^\cap \stackrel{\text{def}}{=} \bigcup_{t^\cap \in [t^\cap]} t^\cap$.

The set of maximal CUTs $\boldsymbol{T}_{max}^\cap$ is then denoted by $\boldsymbol{T}_{max}^\cap$, and defined as follows:

$$\boldsymbol{T}_{max}^\cap = \{\Uparrow t^\cap \mid t^\cap \in \boldsymbol{T}^\cap\} \tag{27}$$

In the rest of this section we will list useful facts about the maximization operator $\Uparrow$ and the set of maximal CUTs $\boldsymbol{T}_{max}^\cap$.

First, the maximization operator preserves the equivalence class. This means that a CUT $t^\cap$ and its maximized CUT $\Uparrow t^\cap$ are both in the same equivalence class set.

Proposition 5.9. *The restriction of the closure operator $\Uparrow$ to any CUTs equivalence class set is an endomorphism of this class, i.e., for all $t^\sqcap \in \mathbf{T}^\sqcap$, if $t^\cap \in t^\sqcap$, then $\Uparrow t^\cap \in t^\sqcap$. Or equivalently, $\forall t^\cap \in \mathbf{T}^\cap, \Uparrow t^\cap \mathrel{\overset{\cap}{\simeq}} t^\cap$.*

Proof. Let $t^\sqcap \in \boldsymbol{T}^\sqcap$, and $t^\cap \in t^\sqcap$. By definition, $\Uparrow t^\cap$ is the union of all CUT in $t^\sqcap$.
We know from proposition 5.7 that $t^\sqcap$ is closed under the union operation, so $\Uparrow t^\cap \in t^\sqcap$. □

Next, all CUTs of a given equivalence class set have the same maximized CUTs.

Proposition 5.10. *The kernel of the closure operator $\Uparrow$ on $\mathbf{T}$ is the equivalence relation $\overset{\cap}{\simeq}$, i.e., if $t_x^\cap \overset{\cap}{\simeq} t_y^\cap$, then $\Uparrow t_x^\cap = \Uparrow t_y^\cap$*

Proof. By definition, if $t_x^\cap \overset{\cap}{\simeq} t_y^\cap$, then $[t_x^\cap] = [t_y^\cap]$, and $\Uparrow t_x^\cap = \Uparrow t_y^\cap$ □

Thus, there is a one-to-one correspondence between the set of a set of equivalence CUTs class sets $\boldsymbol{T}^\sqcap$ and the set of maximal CUTs $\boldsymbol{T}_{max}^\cap$. Moreover, PUTs is a maximal CUT in the sense that among its equivalence class set, it is the unique CUT that has the maximal cardinality.

Proposition 5.11. *The closure operator $\Uparrow$ is order-embedding, i.e.,*

$$t_x^\cap \overset{\cap}{\lesssim} t_y^\cap \Leftrightarrow \Uparrow t_x^\cap \overset{\cap}{\lesssim} \Uparrow t_y^\cap \tag{28}$$

Proof. Let $t_x^\cap, t_y^\cap \in \boldsymbol{T}^\cap$. From proposition 5.9 we know that $t_x^\cap \overset{\cap}{\simeq} \Uparrow t_x^\cap$ and $t_y^\cap \overset{\cap}{\simeq} \Uparrow t_y^\cap$.
$\Rightarrow$: If $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$, then $t_x^\cap \overset{\cap}{\simeq} \Uparrow t_x^\cap \overset{\cap}{\lesssim} \Uparrow t_y^\cap \overset{\cap}{\simeq} t_y^\cap$. So $\Uparrow t_x^\cap \overset{\cap}{\lesssim} \Uparrow t_y^\cap$.
$\Leftarrow$: If $\Uparrow t_x^\cap \overset{\cap}{\lesssim} \Uparrow t_y^\cap$, then $\Uparrow t_x^\cap \overset{\cap}{\simeq} t_x^\cap \overset{\cap}{\lesssim} t_y^\cap \overset{\cap}{\simeq} \Uparrow t_y^\cap$. So $t_x^\cap \overset{\cap}{\lesssim} t_y^\cap$. □

Let $t^\cap \in \boldsymbol{T}^\cap$, we denote $\uparrow t^\cap$ the upper-set of $\boldsymbol{T}$ generated by PUTs in $t^\cap$, i.e.,

$$\uparrow t^\cap \stackrel{\text{def}}{=} \{t \in \boldsymbol{T} \mid \exists t' \in t^\cap, t' \lesssim t\} \tag{29}$$

Proposition 5.12. *The maximization operator $\Uparrow$ is such that all of the following is true:*

- *if $t^\cap \in \mathbf{T}^\cap \setminus \varnothing - \bot^\sqcap$, then $\Uparrow t^\cap = \uparrow t^\cap$;*

- *if* $t^\cap \in \bot^\sqcap$, *then* $\Uparrow t^\cap = \mathbf{T}$;
- $\Uparrow \varnothing = \Uparrow \top^\cap$;

Proof. Let $t^\cap \in \boldsymbol{T}^\cap$:
1) if $t^\cap \in \boldsymbol{T}^\cap \setminus \varnothing - \bot^\sqcap$:
$\subseteq$: Let $t \in \Uparrow t^\cap$. By definition, $\exists t_x^\cap \in [t^\cap], t \in t_x^\cap$. So $t^\cap \stackrel{\sqcap}{\simeq} t_x^\cap$ and $t^\cap \stackrel{\sqcap}{\lesssim} t_x^\cap$.
From proposition 4.5, at least one of the followings holds:
i) $t^\cap \stackrel{\sqcap}{\lesssim} \bot^\cap$ (impossible from our hypothesis);
ii) $t^\cap = \varnothing$ (impossible from our hypothesis);
iii) $\forall t_x \in t_x^\cap, \exists t' \in t^\cap : t' \lesssim t_x$. So for $t \in t_x^\cap$, $\exists t' \in t^\cap : t' \lesssim t$. And thus $t \in \uparrow t^\cap$.
$\supseteq$: Let $t \in \uparrow t^\cap$. Then $\exists t' \in t^\cap, t' \lesssim t$. From lemma 5.8, $t^\cap \cup \{t\} \stackrel{\sqcap}{\simeq} t^\cap$, and thus $t^\cap \cup \{t\} \in [t^\cap]$, and thus $t \in \Uparrow t^\cap$.
2) if $t^\cap \in \bot^\sqcap$, then $\boldsymbol{T} \in [t^\cap]$ and $\Uparrow t^\cap = \boldsymbol{T}$;
3) if $\varnothing \stackrel{\sqcap}{\simeq} \top^\cap$, then from proposition 5.10, $\Uparrow \varnothing = \Uparrow \top^\cap$. □

Figure 3 illustrates the most interesting case in proposition 5.12 which is the case where $t^\cap \notin \bot^\sqcap$ and $t^\cap \neq \varnothing$: black dots represent PUTs of $t^\cap \in \boldsymbol{T}^\cap$, and the fact that a dot is lower than another dot roughly means that the former is more specific than the latter. Circles represent equivalence classes. If $t^\cap$ is a CUT consisting of all the black dots, then the gray zone represents the set of PUTs that belong to $\Uparrow t^\cap$.

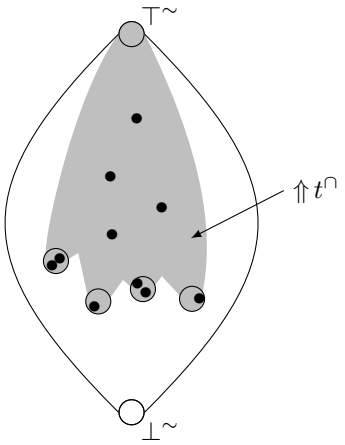


Figure 3: Illustration of a maximal CUT $\Uparrow t^\cap$ with $t^\cap \in \boldsymbol{T}^\cap \setminus \varnothing - \bot^\sqcap$.

A CUT is thus maximal if and only if: either it is the whole set of PUTs, or it is a non-absurd non-empty upper set of PUTs:

Proposition 5.13. *A CUT* $t^\cap$ *is maximal if and only if one of the following is true:*

- $t^\cap = \mathbf{T}$,
- $t^\cap \in \mathbf{T}^\cap \setminus \varnothing - \bot^\sqcap$, *and* $t^\cap$ *is a upper set of* $\mathbf{T}$.

Proof. Let us prove both implications:
$\Rightarrow$: If $t^\cap$ is maximal, then there exists $t^{\cap\prime} \in \boldsymbol{T}^\cap, t^\cap = \Uparrow t^{\cap\prime}$.
i) if $t^{\cap\prime} \in \bot^\sqcap$, then $t^\cap = \boldsymbol{T}$;

ii) if $t^{\cap\prime} = \varnothing$, then $t^\cap = \Uparrow \bot^\cap$, and $\Uparrow \bot^\cap$ is either $\boldsymbol{T}$ or a non-empty upper set of $\boldsymbol{T}$;
iii) if $t^{\cap\prime} \in \boldsymbol{T}^\cap \setminus \varnothing - \bot^\sqcap$, then $t^\cap$ is an non-empty upper set of $\boldsymbol{T}$.

$\Leftarrow$:
i) if $t^\cap = \boldsymbol{T}$, then for all $t^{\cap\prime} \in \bot^\sqcap$, $t^\cap = \Uparrow t^{\cap\prime}$ so $t^\cap$ is maximal;
ii) if $t^\cap \notin \bot^\sqcap$, and $t^\cap$ is a non-empty upper set of $\boldsymbol{T}$, then using propositions 5.10 and 5.9, $\Uparrow t^\cap = t^\cap$, so $t^\cap$ is maximal. □

Remark. Anyways, as $\boldsymbol{T}$ is an upper set of $\boldsymbol{T}$, every maximal CUT is an upper set of $\boldsymbol{T}$.

The last result of this section is the fact the operator $\Uparrow$ defines a closure operator on $\boldsymbol{T}$.

Proposition 5.14. *$\Uparrow$ is a closure operator on $\mathbf{T}$, i.e., it satisfies the following conditions for all CUTs $t^\cap_x, t^\cap_y \in \mathbf{T}^\cap$:*

- $t^\cap_x \subseteq \Uparrow t^\cap_x$, *($\Uparrow$ is extensive);*
- $t^\cap_x \subseteq t^\cap_y \Longrightarrow \Uparrow t^\cap_x \subseteq \Uparrow t^\cap_y$, *($\Uparrow$ is increasing with respect to $\subseteq$);*
- $\Uparrow \Uparrow t^\cap_x = \Uparrow t^\cap_x$, *($\Uparrow$ is idempotent);*

Proof. 1) *Extensive:* Let $t^\cap \in \boldsymbol{T}^\cap$. $t^\cap \in [t^\cap]$ so from definition 5.2, $t^\cap \subseteq \Uparrow t^\cap$, and $\Uparrow$ is extensive.

2) *Increasing w.r.t. $\subseteq$:* Let $t^\cap_x, t^\cap_y \in \boldsymbol{T}^\cap$ such that $t^\cap_x \subseteq t^\cap_y$.
First, we know from proposition 4.6 that $t^\cap_y \overset{\cap}{\lesssim} t^\cap_x$.
i) If $t^\cap_y \in \bot^\sqcap$, then $\Uparrow t^\cap_y = \boldsymbol{T}$ (prop. 5.12) and $\Uparrow t^\cap_x \subseteq \Uparrow t^\cap_y$.
ii) If $t^\cap_x \in \bot^\sqcap$, then so is $t^\cap_y$, and $\Uparrow t^\cap_x = \Uparrow t^\cap_y = \boldsymbol{T}$ (prop. 5.12).
iii) If $t^\cap_y = \varnothing$, then $t^\cap_x = t^\cap_y = \varnothing$, and $\Uparrow t^\cap_x = \Uparrow t^\cap_y$.
iv) If $t^\cap_y \notin \bot^\sqcap$, $t^\cap_y \neq \varnothing$, $t^\cap_x \notin \bot^\sqcap$, and $t^\cap_x \neq \varnothing$: Let $t \in \Uparrow t^\cap_x = \uparrow t^\cap_x$. We know that $\exists t' \in t^\cap_x, t' \lesssim t$. As $t^\cap_x \subseteq t^\cap_y$, then $\exists t' \in t^\cap_y, t' \lesssim t$, and $t' \in \uparrow t^\cap_y = \Uparrow t^\cap_y$.
v) If $t^\cap_x = \varnothing$, then $\Uparrow t^\cap_x = \Uparrow \top^\cap$. $\top$ is a maximal element of $\boldsymbol{T}$, so using lemma 5.8, $t^\cap_y \overset{\cap}{\cong} t^\cap_y \cup \top^\cap$, and from definition 5.2, $\Uparrow t^\cap_y = \Uparrow t^\cap_y \cup \top^\cap$. Using items i) to iv), we know that $\Uparrow t^\cap_x = \Uparrow \top^\cap \subseteq \Uparrow t^\cap_y \cup \top^\cap = \Uparrow t^\cap_y$.
So $\Uparrow t^\cap_x \subseteq \Uparrow t^\cap_y$, and $\Uparrow$ is increasing.

3) *Idempotent:* Let $t^\cap \in \boldsymbol{T}^\cap$.
i) if $t^\cap \in \bot^\sqcap$, then $\Uparrow t^\cap = \boldsymbol{T}$, and as we know that $\boldsymbol{T} \in \bot^\sqcap$, $\Uparrow \Uparrow t^\cap = \Uparrow t^\cap = \boldsymbol{T}$.
ii) if $t^\cap \notin \bot^\sqcap$ and $t^\cap \neq \varnothing$, then:
$\supseteq$: We know from extensivity of $\Uparrow$ that $\Uparrow t^\cap \subseteq \Uparrow \Uparrow t^\cap$.
$\subseteq$: Let $t \in \Uparrow \Uparrow t^\cap$. $\Uparrow t^\cap \overset{\cap}{\cong} t^\cap$ so (prop: 5.7 $\Uparrow t^\cap \notin \bot^\sqcap$, and we also know that $t^\cap \subseteq \Uparrow t^\cap$ so $\Uparrow t^\cap \neq \varnothing$. Using upper-set properties: $\Uparrow \Uparrow t^\cap = \uparrow \Uparrow t^\cap = \uparrow \uparrow t^\cap = \uparrow t^\cap = \Uparrow t^\cap$.
iii) if $t^\cap = \varnothing$, then $\Uparrow t^\cap = \Uparrow \bot^\cap$, and using ii), $\Uparrow t^\cap = \Uparrow \bot^\cap = \Uparrow \Uparrow \bot^\cap = \Uparrow \Uparrow t^\cap$. □

Thus a maximal CUTs $t^\cap_\Uparrow$ is also maximal in the sense that any CUT of its equivalence class is contained in $t^\cap_\Uparrow$.

5.5 Concise CUTs

Concise CUTs will be of special interest in the following of this work. They are those (non-unique) CUTs that have a minimal cardinality in a given equivalence class:

Definition 5.3 (Concise CUT)**.** Let $t^\cap$ be a CUT. $t^\cap$ is said to be *concise* if and only if:

- if $t^\cap$ is absurd, then $t^\cap = \bot^\cap$.
- else, $\forall t \in t^\cap, \nexists t' \in t^\cap$ such that $t' < t$ and $\forall t, t' \in t^\cap$, $t \not\simeq t'$.

Figure 4 below illustrates definition 5.3: black and white dots represent PUTs, and the fact that a dot is lower than another dot roughly means that the former is more specific than the latter. Circles represent equivalence classes. If $t^\cap$ is a CUT consisting of all the black and white dots, then the set of black dots represents a concise CUT that is a subset of $t^\cap$. In the illustrated case, there are $2 \times 1 \times 2 \times 1$ such concise CUT that are subsets of $t^\cap$.

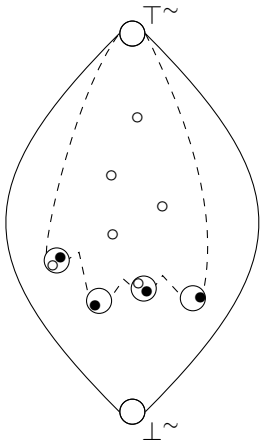


Figure 4: Let the union of black and white dots represent a CUT $t^\cap$. The set of black dots represents a concise CUT that is a subset of $t^\cap$.

6 Implications for the MTT

6.1 A Deep-Semantic Representation Level

As semantic ASymbols are numbers, the pre-order over semantic unit types cannot represent a specialization of meanings. Let us give an example to justify this.

The French semantic unit type ⁽outil⁾ (⁽tool⁾) has an ASlot 1 that corresponds to the person X that uses the tool, and a split ASlot 2 that corresponds either to the activity Y_1 or to the profession Y_2 for which the tool is designed[16]. Now ⁽peigne$_{\mathbf{2a}}$⁾ (⁽comb⁾) has a stricter meaning than ⁽outil⁾, and also an ASlot 2 that now corresponds to the object Y that it is intended to untangle. Thus ⁽peigne$_{\mathbf{2a}}$⁾ cannot be lower than ⁽outil⁾ in the hierarchy of semantic unit types because this would imply that an object is some kind of activity or profession.

We hence propose to introduce a deeper level of representation where one may describe meanings: the *deep semantic level*. We thus establish a distinction between surface and semantic unit types. Let us precise their definition and their actantial structure.

Definition 6.1 (Surface Semantic Unit Types and their ASlots)**.** To every meaningful Lexical Unit Type (LexUT) L is associated a *Surface Semantic Unit Type (SSemUT)* that is denoted ⁽L⁾. The ASlots of ⁽L⁾ correspond to the Semantic Actant Slots (SemASlots) of L as defined in (Mel'čuk, 2004, p.39), and are numbered.

Definition 6.2 (Deep Semantic Unit Types and their ASlots)**.** To every meaningful LexUT L is associated a *Deep Semantic Unit Type (DSemUT)* that is denoted /L\. The set of deep semantic ASymbols are semantic roles (e.g., *agent*, *experiencer*, *object*). The set of ASlots of a DSemUT corresponds to obligatory or optional participants of the linguistic situation denoted by L that are: a) SemASlots of L, or b) SemASlots of a LexUT whose meaning is more generic than that of L.

For instance figure 5 illustrates the actantial structure of /outil\ (⁽An artefact designed for a person X to use it for an activity Y_1 (or for a profession Y_2)⁾) and /ciseaux\, which derives from /tool\. /tool\ has two obligatory actant slots *possessor* and *activity*, and an optional ASlot *profession*. /ciseaux\ inherits the ASlots of /tool\, and restricts the signature of *activity* to be /cut\. As CISEAUX also introduces a SemASlot which is the object to be cut, /ciseaux\ also introduces a new ASlot *objectToBeCut*.

Figure 5: Actantial structures of /outil\ and /ciseaux\.

Actually, one may need to introduce a new ASymbol every time a SemASlot that conveys a new meaning is introduced. The set of semantic roles thus cannot be bound to a small set of universal semantic roles.

[16]See (Mel'čuk, 2004, p.43) for details on split ASlots.

6.2 Refinement of the Semantic Labels Hierarchy

In the RELIEF project, the set of semantic labels are pre-ordered with respect to the specialization of meanings, as is the hierarchy of DSemUT in the UGs framework. We thus propose to identify semantic labels and DSemUTs, and to augment them with actantial structures. One major implication is that one need one DSemUT per meaningful LexUT.

Let us sketch the extension of the scenario described in section **??**. Sophie wants to define the French LexUT PEIGNE$_{\mathbf{2A}}$. She thus needs to characterize its associated DSemUT /peigne$_{\mathbf{2a}}$\. She first opens a new tab in which /peigne$_{\mathbf{2a}}$\ appears in a void box as illustrated in figure 6a. Sophie needs to choose the nearest parent in the hierarchy of DSemUTs. She clicks on the question mark and the current hierarchy of DSemUTs appears like in figure 6b. She chooses /tool\. The box that was void now contains the inherited actantial structure of /tool\ as illustrated in figure 6c. /tool\ has three arbitrarily symbolized ASlots:

- *possessor* for variable X is obligatory and has signature /person\;
- *activity* for variable Y_1 is obligatory and has signature /activity\;
- *profession* for variable Y_2 is optional and has signature /profession\.

Now Sophie may restrict the actantial structure of /peigne$_{\mathbf{2a}}$\.

1. /peigne$_{\mathbf{2a}}$\ is designed to untangle, so Sophie clicks on /activity\ and chooses /untangle\ in the hierarchy of DSemUTs.
2. /peigne$_{\mathbf{2a}}$\ is designed for the weaver profession, so Sophie clicks on /profession\ and chooses /weaver\ in the hierarchy of DSemUTs.
3. the ASlot *profession* is obligatory for /peigne$_{\mathbf{2a}}$\, so Sophie clicks on symbol (⇒), which becomes ⇒.
4. /peigne$_{\mathbf{2a}}$\ introduces a new obligatory ASlot *object* for variable Y with signature /object\. So Sophie clicks on ⊞ New actant slot , and fills a form where she defines a new ASymbol *object*, specifies that this ASlot is obligatory, and specifies the signature: /object\.

Thus the description of the actantial structure of /peigne$_{\mathbf{2a}}$\ looks like in figure 6d.

Let us go back to how lexicographic definitions are currently defined in the RELIEF project. The DSemUT /tool\ has no actantial structure for the moment. Yet,

- the central component of the definition (element CC) specifies the profession for which /peigne$_{\mathbf{2a}}$\ is designed: /weaver\.
- in the peripherical component (element PC), a human reader immediately understands that the activity (PC role *utilisation* here) for which /peigne$_{\mathbf{2a}}$\ is designed is: /peigner$_{\mathbf{2}}$\ (/untangle\).
- /peigner$_{\mathbf{2}}$\ has a SemASlot Y that does not correspond to a participant of /tool\, but that is related to a participant of /untangle\.

Providing /peigne$_{\mathbf{2a}}$\ with an actantial structure that specializes that of /tool\ enables to explicit some of this knowledge, and to give a partial but formal lexicographic definition to /peigne$_{\mathbf{2a}}$\. To complete the formalization of the lexicographic definition of /peigne$_{\mathbf{2a}}$\, one need for instance

$/\text{peigne}_{\mathbf{2a}}\backslash \lesssim ?$
⊞ New actant slot

(a) Starting point to define PEIGNE$_{\mathbf{2A}}$

▽ $/\text{entity}\backslash$ [(⇒)1]

▽ $/\text{physical object}\backslash$ [(⇒)1]

▽ $/\text{artefact}\backslash$ [(⇒)1]

▽ $/\text{instrument}\backslash$ [⇒2]

▷ $/\text{tool}\backslash$ [(⇒)1, ⇒2]

(b) List view of the hierarchy of DSemUTs

$/\text{peigne}_{\mathbf{2a}}\backslash \lesssim /\text{tool}\backslash$
⇒ **possessor :** $/\text{person}\backslash$
⇒ **activity :** $/\text{activity}\backslash$
(⇒) **profession :** $/\text{profession}\backslash$
⊞ New actant slot

(c) Inherited actantial structure of $/\text{tool}\backslash$

$/\text{peigne}_{\mathbf{2a}}\backslash \lesssim /\text{tool}\backslash$
⇒ **possessor :** $/\text{person}\backslash$
⇒ **activity :** $/\text{untangle}\backslash$
⇒ **profession :** $/\text{weaver}\backslash$
⇒ **object :** $/\text{object}\backslash$
⊞ New actant slot

(d) Actantial structure of $/\text{peigne}_{\mathbf{2a}}\backslash$

Figure 6: Definition of the actantial structure of $/\text{peigne}_{\mathbf{2a}}\backslash$.

to represent the fact that the SemASlot X of $/\text{peigne}_{\mathbf{2a}}\backslash$ corresponds to the SemASlot X of $/\text{untangle}\backslash$ for instance.

Now this actantial structure is not suffiscient to represent the lexicographic definition. For instance, $/\text{untangle}\backslash$ has an *agent* ASlot, an this *agent* must correspond to the *possessor* of $/\text{peigne}_{\mathbf{2a}}\backslash$. One thus need UGs to fully represent the definition of $/\text{peigne}_{\mathbf{2a}}\backslash$. Next part is devoted to the definition of UGs.

Part III
Unit Graphs

In the previous part we introduced the types that nodes of any Unit Graph (UG) have, and that specify through slots and signatures how these nodes must be linked to other nodes in the graph. Now is time to introduce the main objects of the Unit Graphs mathematical framework: the Unit Graphs, such as illustrated in figures 7.

7 Unit Graphs (UGs)

Unit types specify how units *must* be linked to other nodes in the UGs, but units *may* also be linked to other units through circumstantial dependencies. Circumstantial dependency symbols are first introduced in section 7.1.

We then introduce the UG support and the UGs themselves (§7.2), a graphical representation for UGs nodes (§7.3), and define cases in which a UG does or does not explicitly comply with the support on which it is defined (§7.4).

7.1 Circumstantial Dependency Symbols Hierarchy

Unit types specify how units are linked to other nodes in the UGs. As for any slot in a predicate, one ASlot of a unit may be filled by only one unit at a time. Now, one may also encounter dependencies of another type in some dependency structures: circumstantial dependencies (Mel'čuk, 2004). Circumstantial relations are considered of type instance-instance contrary to actantial relations. Such dependencies may be optional or multiple (i.e., a unit may govern zero or more other units through relations of a same type). Example of such relations are the deep syntactic representation relations **ATTR**, **COORD**, **APPEND** such as in figure 7b. But we may also use such relations to represent other relations, for example the link between a lexical unit and its associated semantic unit.

We use symbols to distinguish the different kinds of circumstantial dependencies, and thus introduce a finite set of Circumstantial Symbols (CSymbols).

Definition 7.1 (Circumstantial Symbols Set)**.** A CSymbols set is a set of binary relation symbols denoted $\boldsymbol{S_C}$.

CSymbols are often classified in sets and subsets, we thus take into account this need classification and hierarchy, and introduce a partial order over the set of CSymbols.

Definition 7.2 (Pre-order over $\boldsymbol{S_C}$)**.** The CSymbols set is pre-ordered by a relation $\overset{\mathcal{C}}{\lesssim}$, which is induced by a set of *asserted comparisons* $\boldsymbol{C_{S_C}} \subseteq \boldsymbol{T}^{\cap}$. $(\boldsymbol{S_C}, \boldsymbol{C_{S_C}})$ is a directed graph on $\boldsymbol{S_C}$. The pre-order $\overset{\mathcal{C}}{\lesssim}$ is equal to $\boldsymbol{C^*_{S_C}}$ the reflexo-transitive closure of $\boldsymbol{C_{S_C}}$, i.e., $\forall s, s' \in \boldsymbol{S_C}, s' \overset{\mathcal{C}}{\lesssim} s$ iif $(s, s') \in \boldsymbol{C^*_{S_C}}$, i.e., s' is a descendant of s, and s is an ascendant of s'.

Every CSymbol is assigned a signature that specifies what kind of units may be linked trough a relation having this symbol.

Definition 7.3 (Signature of CSymbols)**.** The set of signatures of CSymbols $\{\boldsymbol{\sigma}_s\}_{s \in \boldsymbol{S_C}}$ is a set of couples in $\boldsymbol{T}^{\cap 2}$. For every dependency relation symbol s, $\boldsymbol{\sigma}_s \overset{\text{def}}{=} (domain(s), range(s))$. The set of signatures $\{\boldsymbol{\sigma}_s\}_{s \in \boldsymbol{S_C}}$ must be such that for all $s_1, s_2 \in \boldsymbol{S_C}$ such that $s_1 \overset{\mathcal{C}}{\lesssim} s_2$, $domain(s_1) \overset{\cap}{\lesssim} domain(s_2)$ and $range(s_1) \overset{\cap}{\lesssim} range(s_2)$.

We finally introduce the full hierarchy of CSymbols.

Definition 7.4 (CSymbols hierarchy)**.** A *Circumstantial Symbols (CSymbols) hierarchy* $\mathcal{C} \stackrel{\text{def}}{=} (\boldsymbol{S_C}, \boldsymbol{C_{S_C}}, \mathcal{T}, \{\boldsymbol{\sigma}_s\}_{s\in\boldsymbol{S_C}})$, is composed of:

- $\boldsymbol{S_C}$ a set of Circumstantial Symbols (CSymbols);
- $\boldsymbol{C_{S_C}}$ a set of asserted comparisons;
- $\mathcal{T} = (T_D, \boldsymbol{S_T}, \boldsymbol{\Gamma}, \boldsymbol{\gamma}, \boldsymbol{\Gamma_1}, \boldsymbol{\gamma_1}, \boldsymbol{\Gamma_0}, \boldsymbol{\gamma_0}, C_A, \bot^{\sqcap}_A, \{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}})$ a Conjunctive Unit Types (CUTs) hierarchy;
- $\{\boldsymbol{\sigma}_s\}_{s\in\boldsymbol{S_C}}$ a set of signatures of the CSymbols;

7.2 Definition of UGs

The Unit Graphs (UGs) will enable the description of utterance representation at different representation levels. Parallel with the Conceptual Graphs (CGs), UGs are defined over a so-called support, which is composed of a CUTs hierarchy, a CSymbols hierarchy, and a set of unit identifiers.

Definition 7.5 (Support)**.** A *Support* is a tuple $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{T}, \mathcal{C}, \boldsymbol{M})$ where:

- $\mathcal{T}$ is a *unit types hierarchy*;
- $\mathcal{C}$ is a *Circumstantial Symbols (CSymbols) hierarchy*;
- $\boldsymbol{M}$ is the set of so-called *unit identifiers*.

Let us precise what we mean by unit identifiers. We establish a distinction between:

- units, that are the objects of the represented domain;
- unit identifiers: As units are signs, the set of unit identifiers $\boldsymbol{M}$ are symbols to identify them. Every element of $\boldsymbol{M}$ identifies a specific unit, but multiple elements of $\boldsymbol{M}$ may identify the same unit.
- unit nodes, that are interconnected in UGs and that each represent a unit;
- unit nodes markers, that are chosen in the powerset of $\boldsymbol{M}$ which is denoted $\boldsymbol{M}^{\cap}$: $\boldsymbol{M}^{\cap} \stackrel{\text{def}}{=} 2^{\boldsymbol{M}}$, and that label unit nodes so as to specify what unit each unit node represents.

The UGs enable to represent utterances at different representation levels. Figure 7a illustrates a semantic representation, and figure 7b illustrates a deep syntactic representation. The natural graph representation we use is a finite oriented labelled multigraph, composed of unit nodes, actantial, and circumstantial dependencies, plus a set of asserted unit node equivalences. A unit node is labelled by a couple composed of a CUT that specifies the nature of the unit that is represented, and a unit node marker that enables to identify the represented unit. If the represented unit is unknown, then the unit node may have the generic unit node marker $\varnothing$. On the other hand, if a unit node is marked $\{m_1, m_2\}$, then the unit markers m_1 and m_2 actually identify the same unit. Every dependency arc is labelled by a symbol in $\boldsymbol{S_T} \cup \boldsymbol{S_C}$ that specifies the nature of the link that exists between the unit nodes that this arc connects. Dependency arcs and their symbols are defined in terms of so-called triples. A UG is thus a combination of interconnected unit nodes defined over a given support.

Definition 7.6 (UG)**.** The set of UGs defined over a support $\mathcal{S}$ is denoted $\mathcal{G}(\mathcal{S})$, and each UG $G \in \mathcal{G}(\mathcal{S})$ is a tuple $G \stackrel{\text{def}}{=} (U, \boldsymbol{l}, A, C, Eq)$ where:

- U is the set of *unit nodes*. Every unit node represents a specific unit, but multiple unit nodes may represent the same unit. They are illustrated by rectangles as in figure 7.
- Unit nodes are typed and marked so as to respectively specify what CUT they have and what unit they represent. The marker of a unit node is a set of unit identifiers for mathematical reasons. The set of *unit node markers* is denoted $\boldsymbol{M}^{\cap}$ and is the powerset of $\boldsymbol{M}$. If a unit node is marked by $\varnothing$, it is said to be *generic*, and the represented unit is unknown. On the other hand, if a unit node is marked $\{m_1, m_2\}$, then the unit identifiers m_1 and m_2 actually identify the same unit. $\boldsymbol{l}$ is thus a labelling mapping over U that assigns to each unit node $u \in U$ a couple $\boldsymbol{l}(u) = (t^{\cap}, m^{\cap}) \in \boldsymbol{T}^{\cap} \times \boldsymbol{M}^{\cap}$ of a CUT and a unit node marker. We denote $t^{\cap} = type(u)$ and $m^{\cap} = marker(u)$. Unit nodes are illustrated by rectangles with their label written inside.
- A is the set of *actantial triples* $(u, s, v) \in U \times \boldsymbol{S}_{\mathcal{T}} \times U$. For all $a = (u, s, v) \in A$, the unit represented by v fills the ASlot s of the unit represented by u. We denote $u = governor(a)$, $s = symbol(a)$ and $v = actant(a)$. We also denote $arc(a) = (u, v)$. Actantial triples are illustrated by double arrows.
- C is the set of *circumstantial triples* $(u, s, v) \in U \times \boldsymbol{S}_{\mathcal{C}} \times U$. For all $c = (u, s, v) \in C$, the unit represented by u governs the unit represented by v with respect to s. Conversely, the unit represented by v depends on the unit represented by u with respect to s. We denote $u = governor(c)$, $s = symbol(c)$ and $v = circumstantial(c)$. We also denote $arc(c) = (u, v)$. Circumstantial triples are illustrated by simple arrows.
- $Eq \subseteq U^2$ is the set of so-called *asserted unit node equivalences*. For all $(u_1, u_2) \in U^2$, $(u_1, u_2) \in Eq$ means that u_1 and u_2 represent the same unit. The Eq relation is not an equivalence relation over unit nodes[17]. We thus distinguish explicit and implicit knowledge. Asserted unit node equivalences are illustrated by dashed arrows.
- The *underlying graph of* G denoted $graph(G)$ is a finite, oriented, labelled multigraph where:

  for all $u \in U$, u is a node of $graph(G)$ labelled by $\boldsymbol{l}(u)$;

  for all $a \in A$, $arc(a)$ is an arc of $graph(G)$ labelled by $symbol(a)$;

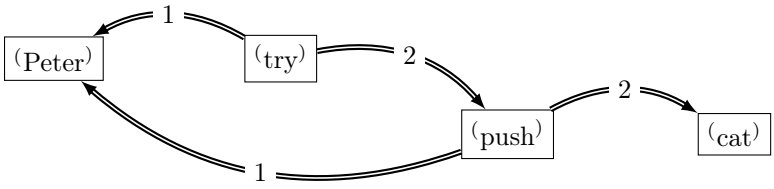
  for all $c \in C$, $arc(c)$ is an arc of $graph(G)$ labelled by $symbol(c)$;

  for all $(u_1, u_2) \in Eq$, (u_1, u_2) is an arc of $graph(G)$ labelled $=$;

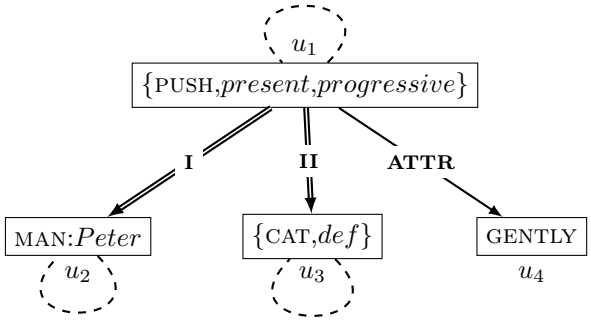
For instance, figure 7a is a semantic representation of sentence *Peter tries to push the cat.* in which units are types by singletons and ASymbols are numbers, in accordance with the MTT. Figure 7b is a simplified deep syntactic representation of *Peter gently pushes the cat.* In this figure unit nodes u_2 and u_4 are typed by singletons, and only unit node u_2 is not generic and has a marker: $\{Peter\}$. A is composed of $(u_1, \mathbf{I}, u_2)$ and $(u_1, \mathbf{II}, u_3)$, where **I** and **II** are ASymbols. C is composed of $(u_1, \mathbf{ATTR}, u_4)$ where **ATTR** is a CSymbol. In the relation Eq there is (u_1, u_1), (u_2, u_2), and (u_3, u_3).

Again, we draw the attention of the reader on the fact that we distinguish units, unit markers, and unit nodes. Several mechanisms may imply that two unit nodes u_1, u_2 represent the same unit, or that two unit markers m_1, m_2 identify the same unit. For UGs as defined above, we identify the following mechanisms:

[17] An equivalent relation is a reflexive, symmetric, and transitive relation.



(a) Semantic representation of sentence *Peter tries to push the cat.*



(b) Deep syntactic representation of sentence *Peter gently pushes the cat.*

Figure 7: Examples of UG.

- u_1, u_2 are asserted to be equivalent, i.e., $(u_1, u_2) \in Eq$. Thus all of the markers in $marker(u_1) \cup marker(u_2)$ identify the same unit.
- as a unit represents a predicate, only one unit may fill each of its ASlot. Hence if (u, r, v_1) and (u, r, v_2) both are in A, then v_1 and v_2 represent the same unit.
- after merging unit nodes that represent the same unit, new couple of unit nodes may appear to represent the same unit according to previous item.

UGs so defined are base objects of the UGs mathematical framework, with which one may formalize among others:

- utterance representations at different representation levels;
- semantic decompositions of lexical units (for lexicographic definitions);
- premises and conclusions of linguistic and grammatical rules;

7.3 Graphical Representation

The graphical representation of a UG is a drawing of the underlying finite, oriented, labelled multigraph. A unit node is represented by a rectangle with its label written inside in the form: *type*":"*marker*", with the following supplementary rules chosen to ease the reading:

- if the type or the maker is a singleton, brackets may be avoided;
- if the marker is the generic marker (the empty set), it may be avoided along with its semi-column;
- if the type is the empty set, it may be avoided, but the semi-column must be written;
- if the type and the marker are the empty sets, there may be nothing written in the nodes.

Figure 8 below illustrates these rules. All the unit nodes of each line share the same label.

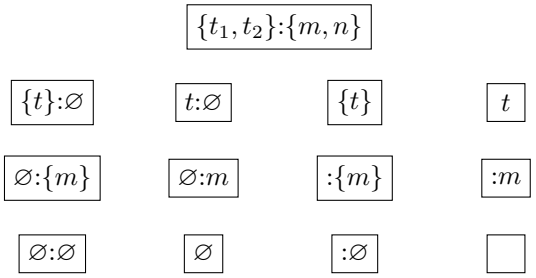


Figure 8: Graphical Representation of unit nodes and their labels:
All the unit nodes of each line share the same label.

We also use the following conventions to distinguish the different relations:

- actantial relations are drawn with a double arrow.
- circumstantial dependencies are drawn with a single arrow.
- *Eq* relations are drawn with a dashed line. The label may not be written. If the relation is symmetric, then the arc is non-oriented.

Figure 9 below illustrates these conventions.

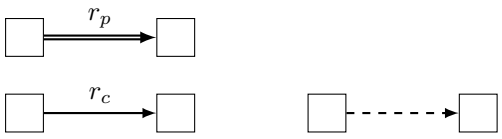


Figure 9: Graphical Representation of arcs and their labels. Top left: a actantial triple; Bottom left: a circumstantial triple; right: an asserted unit nodes equivalence.

7.4 Explicit Support Compliance

In a UG G, dependency relations and unit nodes may be explicitly compliant with the support over which G is defined, by respecting ASlots or signatures for instance.

Let us underline that if a unit node or a dependency relation is not explicitly compliant with the support, this is not necessarily wrong: it may only means that some knowledge is not explicit. We thus make the open-world assumption, which means that a UG along with the support on which it is defined represents explicit knowledge, and that additional knowledge may be inferred. As we shall see, this assumption eases the compatibility between the UGs mathematical framework and the Semantic Web Formalisms (SWFs).

First, any actantial relation incident from a unit node should represent a non-prohibited ASlot of the type of this unit node.

Definition 7.7 (Explicit support compliance of a unit node)**.** In a UG $(U, \boldsymbol{l}, A, C, Eq) \in \mathcal{G}(\mathcal{S})$, a unit node u is said to *explicitly comply with the support* $\mathcal{S}$ if and only if: for all $a \in A$ such that $governor(a) = u$, $symbol(a) \in \boldsymbol{\alpha}^{\cap}(type(u)) - \boldsymbol{\alpha}^{\cap}_{\mathbf{0}}(type(u))$.

Then, for an actantial triple, the actant symbol must be a non-prohibited ASlot of the governor's type, and the participant's type should explicitly comply with the signature.

Definition 7.8 (Explicit support compliance of a participation triple)**.** In a UG $(U, \boldsymbol{l}, A, C, Eq) \in \mathcal{G}(\mathcal{S})$, a participation triple $(u, r, v) \in A$ is said to *explicitly comply with the support* $\mathcal{S}$ if and only if: $r \in \boldsymbol{\alpha}^{\cap}(type(u)) - \boldsymbol{\alpha}^{\cap}_{\mathbf{0}}(type(u))$ and $\boldsymbol{\varsigma}^{\cap}_{type(u)}(r) \subseteq type(v)$.

Finally for a circumstantial triple, the type of the governor and the dependent should explicitly comply with the signature of the CSymbol.

Definition 7.9 (Explicit support compliance of a circumstantial triple)**.** In a UG $(U, \boldsymbol{l}, A, C, Eq) \in \mathcal{G}(\mathcal{S})$, a circumstantial triple $(u, r, v) \in C$ is said to *explicitly comply with the support* $\mathcal{S}$ if and only if: $domain(r) \subseteq type(u)$ and $range(r) \subseteq type(v)$.

A UG is thus explicitly compliant with the support on which it is defined if and only if all of its dependency triples are explicitly compliant on the support.

Definition 7.10 (Explicit support compliance of a UG)**.** A UG $G \in \mathcal{G}(\mathcal{S})$ is said to *explicitly comply with the support* $\mathcal{S}$ if and only if all of its dependency arcs $d \in A \cup C$ is explicitly compliant with the $\mathcal{S}$.

8 Mappings of UGs

UGs have an underlying oriented labelled graph. It is thus convenient for reasoning applications to introduce the notion of UGs homomorphism. Recall that for non-labelled graphs, an homomorphism from H to G, is an edge-preserving mapping from nodes of H to nodes of G. We will thus introduce the notion of homomorphism of UGs, based on homomorphism of their underlying oriented labelled graphs. Let us first introduce the notion of UGs mapping. A UGs mapping corresponds to a mapping of their underlying graphs.

Let $H = (U^h, \boldsymbol{l}^h, A^h, C^h, Eq^h)$ and $G = (U^g, \boldsymbol{l}^g, A^g, C^g, Eq^g)$ be two UGs defined over the same support.

Definition 8.1. A UGs mapping f from H to G, written $f : H \to G$, corresponds to a mapping of their underlying oriented labelled graphs, i.e., a mapping $f : U^h \to U^g$ from the unit nodes of H to the unit nodes of G.

8.1 Homomorphism

There is a homomorphism of UGs if there is a homomorphism on their underlying oriented labelled multigraph. To define such a homomorphism, one needs to choose pre-orders over labels for unit nodes and arcs. We define the homomorphism using the following set of pre-orders:

- inclusion for markers;
- pre-order $\overset{\cap}{\lesssim}$ for types;
- equality for actantial triples;
- pre-order $\overset{c}{\lesssim}$ for circumstantial triples;
- equality for asserted equivalence relations.

Definition 8.2. There is a *homomorphism* from H to G if and only if there exists a mapping $\pi : H \to G$ such that all of the following is true:

- $\forall u \in U^h$, $marker(u) \subseteq marker(\pi(u))$;
- $\forall u \in U^h$, $type(\pi(u)) \overset{\cap}{\lesssim} type(u)$;
- $(u, s, v) \in A^h \Rightarrow (\pi(u), s, \pi(v)) \in A^g$;
- $(u, s, v) \in C^h \Rightarrow \exists c \in C^g, arc(c) = (\pi(u), \pi(v))$ and $symbol(c) \overset{c}{\lesssim} s$;
- $(u, v) \in Eq^h \Rightarrow (\pi(u), \pi(v)) \in Eq^g$.

For example, figure 10 illustrates a homomorphism from H to G.

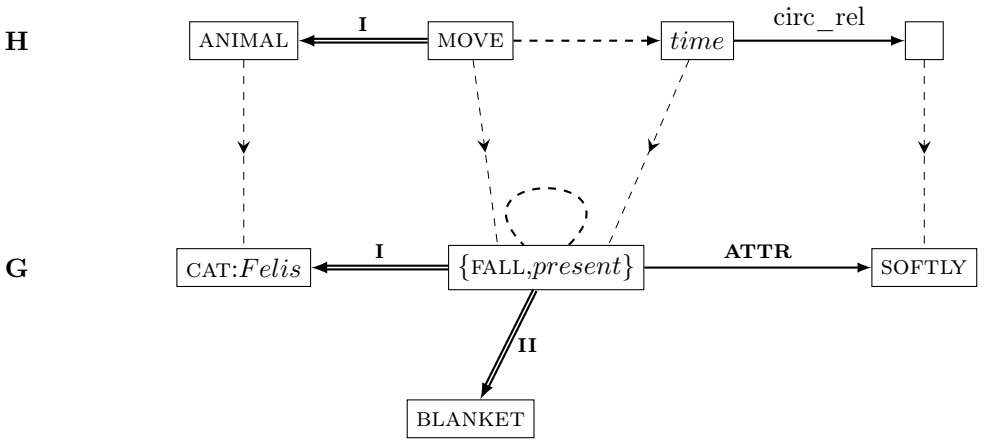


Figure 10: Illustration of a homomorphism from H to G.

8.2 Hom-Equivalence

Two UGs are hom-equivalent if there exists a homomorphism from one to the other and vice versa.

Definition 8.3 (Hom-Equivalence)**.** Let $G = (U^g, \mathbf{l}^g, A^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S})$ and $H = (U^h, \mathbf{l}^h, A^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S})$ be two UGs defined on the same support. G and H are Hom-Equivalent if there exists a homomorphism from G to H and a homomorphism from H to G.

Figure 11 below illustrates two hom-equivalent UGs that are different.

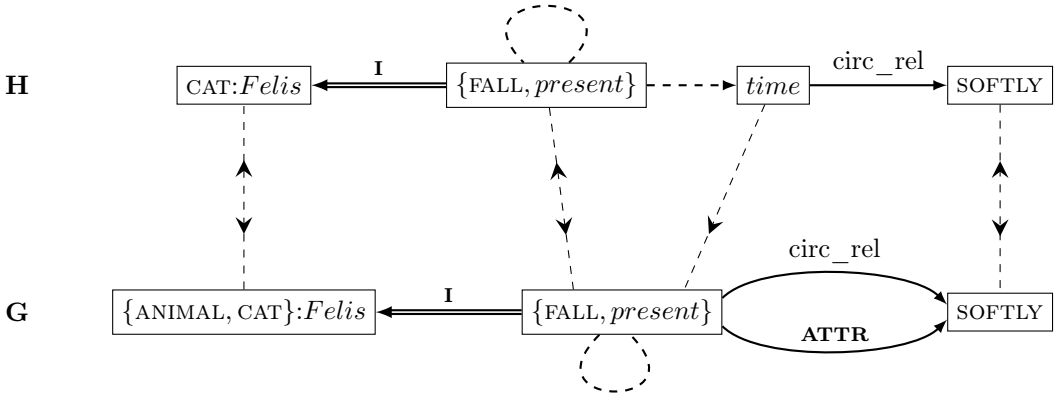


Figure 11: Illustration of a hom-equivalence between H and G.

8.3 Isomorphism

A hom-equivalence is an isomorphism if the two mappings are the inverse one of another.

Definition 8.4 (Isomorphism)**.** Let $G = (U^g, \boldsymbol{l}^g, A^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S})$ and $H = (U^h, \boldsymbol{l}^h, A^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S})$ be two UGs defined on the same support. A mapping $\pi : G \to H$ is a *isomomorphism* if and only if it is a bijective homomorphism.

Figure 12 below illustrates two isomorphic UGs that are different.

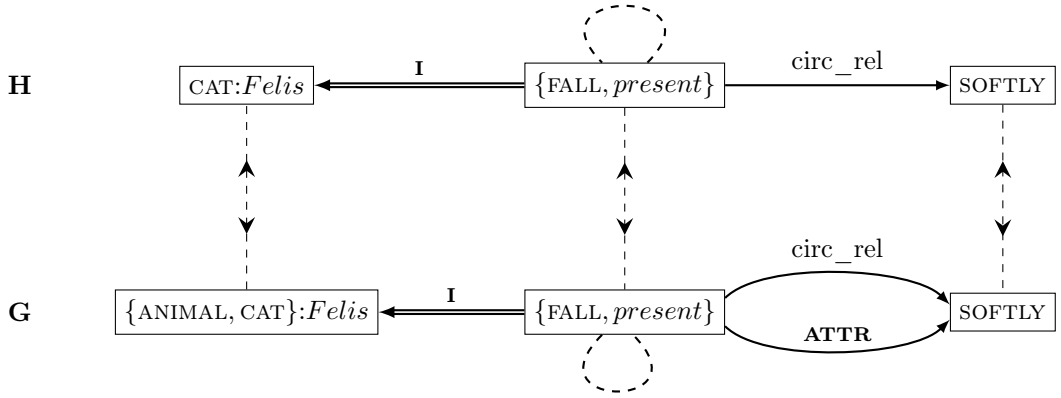


Figure 12: Illustration of a isomorphism between H and G.

9 Homomorphisms-based Semantics of UGs

9.1 Closure of a UG

The UGs framework makes the open-world assumption, which means that a UG along with the support on which it is defined represents explicit knowledge, and that additional knowledge may be inferred. Consider the UG $G = (U, \boldsymbol{l}, A, C, Eq)$ defined over the support $\mathcal{S}$ illustrated in figure 13a. Some knowledge in G is implicit:

1. two unit nodes u_1 and u_2 share a common unit marker $Mary$, so one may infer that they represent the same unit. (u_1, u_2) may be added to Eq.
2. every PUT is a subtype of $\top$, so one could add $\top$ to all the types of unit nodes in G.
3. there are two unit nodes v_1 and v_2 that fill the same ASlot *activity* of the unit node typed $^{/}\text{instrument}^{\backslash}$. So one may infer that v_1 and v_2 represent the same unit. Said otherwise, (v_1, v_2) may be added to Eq.

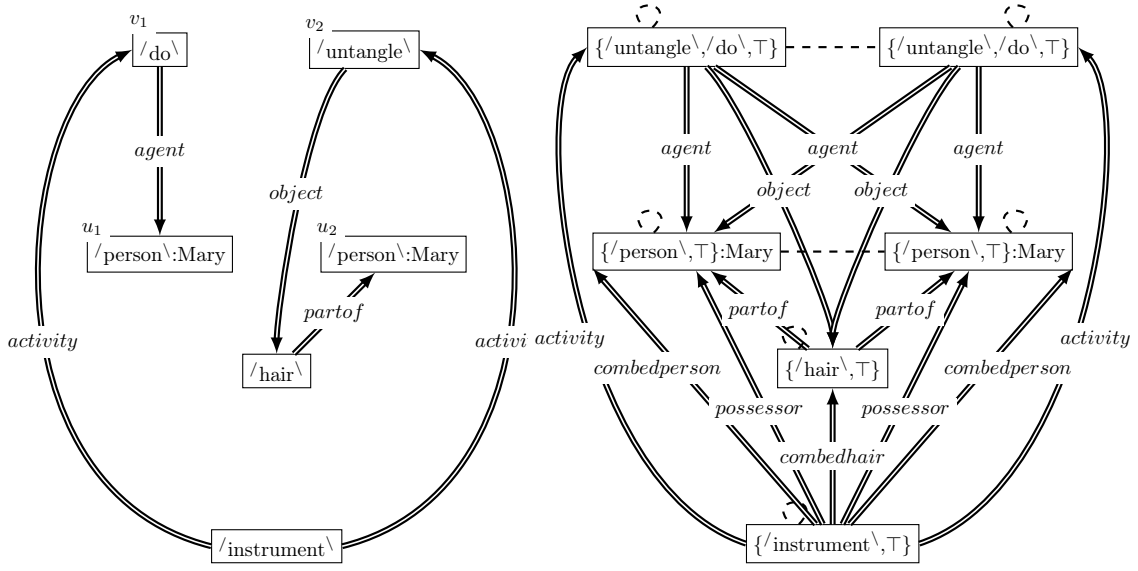
Each of the rules behind these cases explicit knowledge in G. More generally, table 1 lists a set of rules that one may use to explicit knowledge in any UG. Cases 1 to 3 respectively correspond to rules **mrk-eq**, **u-typ**, **a-fp**. The complete set of rules defines the axiomatization of the UGs semantics.

Definition 9.1 (Closing a UG)**.** The process of applying the set of rules of figure 1 on G until none of them has any effect is called *closing* G, and results in $cl(G)$.

Figure 13b illustrates the closure of G, where all of the inferable knowledge has been made explicit.

u-typ	For all $u \in U$, and $type(u) \overset{\cap}{\lesssim} t^{\cap}$	Add $t^{\cap}$	in $type(u)$
u-bot	For all $u \in U$, if $\bot \in type(u)$,	Error: inconsistency !	
eq-ref	For all $u \in U$	Add (u, u)	in Eq
eq-sym	For all $(u_1, u_2) \in Eq$	Add (u_2, u_1)	in Eq
eq-trans	For all (u_1, u_2) and $(u_2, u_3) \in Eq$	Add (u_1, u_3)	in Eq
eq-typ	For all $(u_1, u_2) \in Eq$	Add $type(u_1)$	in $type(u_2)$
eq-mrk	For all $(u_1, u_2) \in Eq$	Add $marker(u_1)$	in $marker(u_2)$
mrk-eq	For all $u_1, u_2 \in U$, if $marker(u_1) \cap marker(u_2) \neq \varnothing$,	Add (u_1, u_2)	in Eq
a-eq-g	For all $(u_1, s, v) \in A$ and $(u_1, u_2) \in Eq$	Add (u_2, s, v)	in A
a-eq-a	For all $(u, s, v_1) \in A$ and $(v_1, v_2) \in Eq$	Add (u, s, v_2)	in A
a-fp	For all (u, s, v_1) and $(u, s, v_2) \in A$	Add (v_1, v_2)	in Eq
a-radix	For all $(u, s, v) \in A$	Add $\boldsymbol{\gamma}(s)$	in $type(u)$
a-obl	For all $u \in U$ and $s \in \boldsymbol{S_{\mathcal{T}}}$, if $\boldsymbol{\gamma_1}(s) \in type(u)$	Ensure there exists (u, s, v) in A	
a-pro	For all $(u, s, v) \in A$, if $\boldsymbol{\gamma_0}(s) \in type(u)$,	Error: inconsistency !	
a-sig	For all $(u, s, v) \in A$	Add $\boldsymbol{\varsigma}^{\cap}_{type(u)}(s)$	in $type(v)$
c-eq-g	For all $(u_1, s, v) \in C$ and $(u_1, u_2) \in Eq$	Add (u_2, s, v)	in C
c-eq-c	For all $(u, s, v_1) \in C$ and $(v_1, v_2) \in Eq$	Add (u, s, v_2)	in C
c-dom	For all $(u, s, v) \in C$	Add $domain(s)$	in $type(u)$
c-rng	For all $(u, s, v) \in C$	Add $range(s)$	in $type(v)$
c-sop	For all $(u, s_1, v) \in C$ and $s_1 \overset{c}{\lesssim} s_2$	Add (u, s_2, v)	in C

Table 1: Semantics of the Unit Graphs.



(a) Incomplete deep semantic representation G (b) Closure of the unit graph G

Figure 13: Closure of a UG.

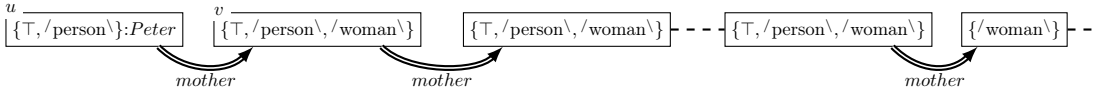


Figure 14: Illustration of an infinite closure of a simple Unit Graph

9.2 Reasoning with Homomorphisms

Now that we provided UGs with semantics and that we have means to explicit knowledge, we will define the prime decision problem of the UGs framework: *Considering two UGs G and H defined over the same support $\mathcal{S}$, does the knowledge of G entails the knowledge of H ?* The notion of *entailment* may intuitively be defined for UGs as follows: G entails H if and only if $cl(G)$ *includes* H.

There are two issues with this definition of entailment. The first is that one needs to define what is the precise meaning of *inclusion of a UG*. The second is that $cl(G)$ may be infinite, thus preventing decidability of entailment.

The answer to the first issue is straightforward as we already defined the UGs homomorphism. We may thus formally define the entailment:

Definition 9.2 (Entailment and equivalence)**.** Let H and G be two UGs defined over a support $\mathcal{S}$.

- G *entails* H, or H is a *semantic consequence* of G, noted $G \vDash_{\mathrm{h}} H$, if and only if there exists a homomorphism from H to $cl(G)$.
- H and G are *equivalent*, noted $H \equiv_{\mathrm{h}} G$, if and only if $H \vDash_{\mathrm{h}} G$ and $G \vDash_{\mathrm{h}} H$.

Now, the second issue is more problematic. Indeed, the closure of a finite UG may be infinite, thus preventing decidability of the decision problem. This problem is illustrated on a simple example in figure 14, suppose one asserts that deep semantic PUTs $^{/}\mathrm{person}^{\backslash}$ has a obligatory ASlot *mother* with $\boldsymbol{\varsigma}_{^{/}\mathrm{person}^{\backslash}}(mother) = {}^{/}\mathrm{woman}^{\backslash}$, and of course, $^{/}\mathrm{woman}^{\backslash} \lesssim {}^{/}\mathrm{person}^{\backslash}$. Consider the UG $G = (\{u\}, \boldsymbol{l}, \varnothing, \varnothing, \{(u,u)\})$, such that $marker(u) = Peter$ and $type(u) = \{\top, {}^{/}\mathrm{person}^{\backslash}\}$. One knows that the unit represented by u should have a unit of type $^{/}\mathrm{woman}^{\backslash}$ that fills its obligatory ASlot *mother*. So rule **a-obl** is applicable and one could add a unit node v to represent that argument, with $(u, mother, v) \in A$. Rule **u-typ** will then make v be of type $\{\top, {}^{/}\mathrm{person}^{\backslash}, {}^{/}\mathrm{woman}^{\backslash}\}$, and rule **a-obl** is again applicable on v. Thus $cl(G)$ is an infinite chain of unit nodes having type $\{\top, {}^{/}\mathrm{person}^{\backslash}, {}^{/}\mathrm{woman}^{\backslash}\}$ and that fill the ASlot *mother* of one another.

In the set of inference rules of table 1, only rule **a-obl** adds unit nodes to the UG when triggered. An open problem is thus to define a sufficient condition on the unit types hierarchy so that we are ensured that the closure of a finite UG is finite.

10 Model Semantics for UGs

10.1 Model of a Support

In this section we will provide the UGs framework with a model semantic based on a relational algebra. Let us first introduce the definition of the model of a support.

Definition 10.1 (Model of a support)**.** Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \boldsymbol{M})$ be a support. A model of $\mathcal{S}$ is a couple $M = (D, \delta)$. D is a set called the domain of M that contains a special element denoted $\bullet$ that represents *nothing*, plus at least one other element. δ is denoted the *interpretation function* and must be such that:

- M is a model of $\mathcal{T}$;
- M is a model of $\mathcal{C}$;
- $\forall m \in \boldsymbol{M}, \delta(m) \in D \setminus \bullet$;

This definition requires the notion of model of a unit types hierarchy, and model of a CSymbols hierarchy. We will sequentially introduce these notions in the following sections.

10.2 Model of a Hierarchy of Unit Types

The interpretation function δ associates with any primitive unit type $t \in \boldsymbol{T}$ a relation $\delta(\{t\})$ of arity $1 + valency(t)$ with the following set of attributes (eq. 30):

- a primary attribute denoted 0 ($0 \notin \boldsymbol{S}_{\mathcal{T}}$) that provides $t^{\cap}$ with the semantics of a class;
- an attribute for each of its ASlot in $\boldsymbol{\alpha}^{\cap}(t^{\cap})$ that provide $t^{\cap}$ with the dual semantics of a relation.

$$\begin{gathered} \forall t \in \boldsymbol{T}, \delta(\{t\}) \subseteq D^{1+valency(t)} \\ \text{with attributes } \{0\} \cup \boldsymbol{\alpha}(t) \end{gathered} \tag{30}$$

Every tuple r of $\delta(t^{\cap})$ can be identified to a mapping, still denoted r, from the attribute set $\{0\} \cup \boldsymbol{\alpha}^{\cap}(t^{\cap})$ to the universe D. r describes how a unit of type $t^{\cap}$ is linked to its actants. $r(0)$ is the unit itself, and for all $s \in \boldsymbol{\alpha}^{\cap}(t^{\cap})$, $r(s)$ is the unit that fills ASlot s of $r(0)$. If $r(s) = \bullet$, then *there is no unit* that fills ASlot s of $r(0)$. A given unit may be described at most once in $\delta(t^{\cap})$, so 0 is a unique key in the interpretation of every PUT:

$$\forall t \in \boldsymbol{T}, \forall r_1, r_2 \in \delta(\{t\}), \quad r_1(0) = r_2(0) \Rightarrow r_1 = r_2 \tag{31}$$

$\top$ must be the type of every unit, except for the special *nothing* element $\bullet$, and $\bot$ must be the type of no unit. As the projection $\pi_0\delta(\{t\})$ on the main attribute 0 represents the set of units having type t, equations 32 and 33 model these restrictions.

$$\pi_0\delta(\{\top\}) = D \setminus \bullet; \tag{32}$$

$$\delta(\{\bot\}) = \varnothing \tag{33}$$

The ASlot s of the obligat $\boldsymbol{\gamma_1}(s)$ must be filled by some unit, but no unit may fill ASlot s of the prohibet $\boldsymbol{\gamma_0}(s)$. As for every $s \in \boldsymbol{\alpha}(t)$, the projection $\pi_s\delta(\{t\})$ represents the set of units that fill the ASlot s of some unit that has type t, equations 34 and 35 model these restrictions.

$$\forall s \in \boldsymbol{S_\mathcal{T}}, \bullet \notin \pi_s\delta(\{\boldsymbol{\gamma_1}(s)\}); \tag{34}$$
$$\forall s \in \boldsymbol{S_\mathcal{T}}, \pi_s\delta(\{\boldsymbol{\gamma_0}(s)\}) = \{\bullet\}; \tag{35}$$

For every unit of PUT t and for every ASlot of t, the unit that fills ASlot s must be either *nothing*, or a unit of type $\boldsymbol{\varsigma}_t(s)$:

$$\forall t \in \boldsymbol{T}, \forall s \in \boldsymbol{\alpha}(t), \pi_s\delta(\{t\}) \setminus \bullet \subseteq \pi_0\delta(\boldsymbol{\varsigma}_t(s)) \tag{36}$$

Now if a unit $i \in D$ is of type t_1 and t_1 is more specific than t_2, then the unit is also of type t_2, and the description of i in $\delta(\{t_2\})$ must correspond to the description of i in $\delta(\{t_1\})$. Equivalently, the projection of $\delta(\{t_1\})$ on the attributes of $\delta(\{t_2\})$ must be a sub-relation of $\delta(\{t_2\})$:

$$\forall t_1 \lesssim t_2, \pi_{\{0\}\cup\boldsymbol{\alpha}(t_2)}\delta(\{t_1\}) \subseteq \delta(\{t_2\}) \tag{37}$$

The interpretation of a CUT is the join of the interpretation of its constituting PUTs, except for $\varnothing$ which has the same interpretation as $\{\top\}$, and asserted absurd CUT $t^\cap \in \bot_A^\sqcap$ that contain no unit.

$$\forall t^\cap \in \boldsymbol{T}^\cap \setminus \varnothing - \bot_A^\sqcap, \delta(t^\cap) = \bowtie_{t \in t^\cap} \delta(\{t\}) \tag{38}$$
$$\delta(\varnothing) = \delta(\{\top\}) \tag{39}$$
$$\forall t^\cap \in \bot_A^\sqcap, \delta(t^\cap) = \varnothing \tag{40}$$

We may now define the model of a unit type hierarchy.

Definition 10.2. Let be a CUTs hierarchy
$\mathcal{T} = (T_D, \boldsymbol{S_\mathcal{T}}, \boldsymbol{\Gamma}, \boldsymbol{\gamma}, \boldsymbol{\Gamma_1}, \boldsymbol{\gamma_1}, \boldsymbol{\Gamma_0}, \boldsymbol{\gamma_0}, C_A, \bot_A^\sqcap, \{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}})$. A model of $\mathcal{T}$ is a couple $M = (D, \delta)$ such that the interpretation function δ satisfies equations 30 to 40.

10.3 Model of a Hierarchy of Circumstantial Symbols

So as to be also a model of a circumstantial dependency symbols hierarchy, the interpretation function δ must be extended and further restricted as follows.

The interpretation function δ associates with every CSymbol $s \in \boldsymbol{S_\mathcal{C}}$ a binary relation $\delta(s)$ with two attributes : *gov* which stands for governor, and *circ* which stands for circumstantial.

$$\forall s \in \boldsymbol{S_\mathcal{C}}, \delta(s) \subseteq (D \setminus \bullet)^2, \text{ a relation with attributes } \{gov, circ\}; \tag{41}$$

Parallel with binary relations in the semantic model of the CGs formalism, if a CSymbol s_1 is more specific than another CSymbol s_2, then the interpretation of s_1 must be included in the interpretation of s_2.

$$\forall s_1, s_2 \in \boldsymbol{S_\mathcal{C}}, s_1 \overset{c}{\lesssim} s_2 \Rightarrow \delta(s_1) \subseteq \delta(s_2) \tag{42}$$

Finally, the type of the units that are linked through a CSymbol s must correspond to the signature of s.

$$\forall s \in \boldsymbol{S_C}, \pi_{gov}\delta(s) \subseteq \pi_0\delta(domain(s)); \tag{43}$$

$$\forall s \in \boldsymbol{S_C}, \pi_{circ}\delta(s) \subseteq \pi_0\delta(range(s)); \tag{44}$$

We may thus define the model of a CSymbols hierarchy.

Definition 10.3 (Model of a Circumstantial Dependency Symbols Hierarchy)**.** Let be a CSymbols hierarchy $\mathcal{C} = (\boldsymbol{S_C}, \boldsymbol{C_{S_C}}, \mathcal{T}, \{\boldsymbol{\sigma}_s\}_{s\in\boldsymbol{S_C}})$. A model of $\mathcal{C}$ is a model $M = (D, \delta)$ of $\mathcal{T}$ such that the interpretation function δ satisfies equations 41 to 44.

10.4 Model Satisfying a UG and Semantic Consequence

Now that the model of a support is fully defined, we may define the model of a UG. A model of a UG is a model of the support on which it is defined, augmented with an *assignment* mapping over unit nodes that assigns to every unit node an element of D.

Definition 10.4 (Model of a UG)**.** Let $G = (U, \boldsymbol{l}, A, C, Eq)$ be a UG defined over a support $\mathcal{S}$. A model of G is a triple (D, δ, β) where:

- (D, δ) is a model of $\mathcal{S}$;
- β, called an *assignment*, is a mapping from U to D.

So as to satisfy the UG, the assignment β must satisfy a set of requirements. First, if a unit node $u \in U$ has a marker $m \in marker(u)$, then the assignment of u must correspond to the interpretation of m.

$$\forall u \in U, \forall m \in marker(u), \beta(u) = \delta(m) \tag{45}$$

Then, the assignment of any unit node u must belong to the set of units that have type $type(u)$.

$$\forall u \in U, \beta(u) \in \pi_0\delta(type(u)) \tag{46}$$

For every actantial triple $(u, s, v) \in A$, and as $\{\boldsymbol{\gamma}(s)\}$ is the CUT that introduces a ASlot s, the interpretation $\delta(\{\boldsymbol{\gamma}(s)\})$ must reflect the fact that the unit represented by v fills the actant slot s of the unit represented by u.

$$\forall (u, s, v) \in A, \pi_{0,s}\delta(\{\boldsymbol{\gamma}(s)\}) = \{(\beta(u), \beta(v))\} \tag{47}$$

Similarly, for every circumstantial triple $(u, s, v) \in C$, the interpretation of s must reveal the fact that the unit represented by v depends on the unit represented by u with respect to s.

$$\forall (u, s, v) \in C, (\beta(u), \beta(v)) \in \delta(s) \tag{48}$$

Finally, if two unit nodes are asserted to be equivalent, then the unit they represent are the same and their assignment must be the same.

$$\forall (u_1, u_2) \in Eq, \beta(u_1) = \beta(u_2) \tag{49}$$

We may now define the notion of satisfaction of a UG by a model.

Definition 10.5 (Model satisfying a UG)**.** Let $G = (U, \boldsymbol{l}, A, C, Eq)$ be a UG defined over a support $\mathcal{S}$, and (D, δ, β) be a model of G. (D, δ, β) is a *model satisfying* G, noted $(D, \delta, \beta) \vDash_{\mathrm{m}} G$, if β is an assignment that satisfies equations 45 to 49.

Using the notion of a support model and a UG model it is possible to define an entailment relation between UGs as follows.

Definition 10.6 (Entailment and equivalence)**.** Let H and G be two UGs defined over a support $\mathcal{S}$.

- G *entails* H, or H is a *semantic consequence* of G, noted $G \vDash_{\mathrm{m}} H$, if and only if for any model (D, δ) of $\mathcal{S}$ and for any assignment β_G such that $(D, \delta, \beta_G) \vDash_{\mathrm{m}} G$, then there exists an assignment β_H of the unit nodes in H such that $(D, \delta, \beta_H) \vDash_{\mathrm{m}} H$.
- H and G are *model-equivalent*, noted $H \equiv_{\mathrm{m}} G$, if and only if $H \vDash_{\mathrm{m}} G$ and $G \vDash_{\mathrm{m}} H$.

The such defined model semantics of the UGs corresponds actually to a meta-schema of database. Informations carried by a single UG correspond to some informations in the database, and we will need algorithms to explicit knowledge in the database. Let H and G be two UGs defined over a support $\mathcal{S}$. One open question is: $G \vDash_{\mathrm{h}} H \Leftrightarrow G \vDash_{\mathrm{m}} H$?

Part IV
Rules and Definitions

11 Rules

11.1 Definition of Rules

One question one may ask at this point is: how to represent the correspondence between the actantial structure of a DSemUT, and the actantial structure of its associated SSemUT? First and parallel with CGs, we will define generic unit nodes and λ-UGs that enable to distinguish some generic unit nodes of a UG.

Definition 11.1 (Genericity)**.** In a UG $G \in \mathcal{G}(\mathcal{S})$, a unit node u is said to be *generic* if and only if $marker(u) = \varnothing$. G is said to be *generic* if and only if all of its unit nodes are generic.

Definition 11.2 (λ-UG and Freedom)**.** A *λ-UG* $L = \{u_1, \ldots, u_n\}G$ defined over a support $\mathcal{S}$, is composed of a UG $G = (U, \boldsymbol{l}, A, C, Eq) \in \mathcal{G}(\mathcal{S})$, and a set of generic unit nodes of G, $\{u_1, \ldots, u_n\}$, denoted the free nodes of L. A unit node $u \in U$ is said to be *free* if and only if $u \in \{u_1, \ldots, u_n\}$. G is said to be *free* if and only if all of its unit nodes are free, i.e., if $\{u_1, \ldots, u_n\} = U$. n is denoted the *size* of G.

λ-UGs are actually generalized UGs, and a UG may be considered as a λ-UG of size 0.

Definition 11.3 (Merge of two λ-UG with respect to a partial function)**.** Let be two λ-UG defined over a support $\mathcal{S}$:

- $C = \{u_1^c, \ldots, u_n^c\}(U^c, \boldsymbol{l}^c, A^c, C^c, Eq^c)$;
- $G = \{u_1^g, \ldots, u_m^g\}(U^g, \boldsymbol{l}^g, A^g, C^g, Eq^g)$.

Let η be a partial function from $\{u_1^c, \ldots, u_n^c\}$ to U^g. The merge of C and G with respect to η is denoted $merge(G, C, \eta)$ and is obtained as follows:

1. add C and G;
2. for all $(u^c, u^g) \in \eta$, merge u^c and u_g. The resulting node is free if $u_g \in \{u_1^g, \ldots, u_m^g\}$, not free otherwise.

λ-UG are actually generalized UGs, and a UG may be considered as a λ-UG of size 0. A rule may then be simply represented by two λ-UGs and a bijection between their free nodes.

Definition 11.4 (Rule)**.** A *Rule* defined over a support $\mathcal{S}$ is a triple $R \stackrel{\text{def}}{=} (H, C, \kappa)$ where:

- $H = \{u_1^h, \ldots, u_n^h\}H'$ is a λ-UG defined over $\mathcal{S}$ called the hypothesis;
- $C = \{u_1^c, \ldots, u_n^c\}C'$ is a λ-UG defined over $\mathcal{S}$ called the conclusion;
- κ is a bijection from $\{u_1^h, \ldots, u_n^h\}$ to $\{u_1^c, \ldots, u_n^c\}$.

A rule is said to be *applicable* to a UG G if and only if there exists a homomorphism from H to G.

Definition 11.5 (Applicability of a Rule)**.** Let a rule $R = (H, C, \kappa)$ and a UG G be defined over a support $\mathcal{S}$. R is *applicable* to G if and only if there exists a homomorphism from H to G.

Let π be such a homomorphism from H to G. The application of R on G with respect to π is the UG obtained by merging C in G with respect to $\pi \circ \kappa^{-1}$.

Definition 11.6 (Application of a Rule)**.** Let $R = (H, C, \kappa)$ be a rule applicable to a UG G, and let π be a homomorphism from H to G. The application of R on G with respect to π is the UG obtained by merging C in G with respect to $\pi \circ \kappa^{-1}$, i.e.,

1. add C to G;
2. for all $(u^c, u^g) \in \pi \circ \kappa^{-1}$, merge u^c and u^g as follows: (i) add a new node u, with $type(u) = type(u^c) \cup type(u^g)$ and $marker(u) = marker(u^c) \cup marker(u^g)$; (ii) replace u^c and u^g by u in any dependency triple in $A \cup C$ and in any element of Eq.

Among other, rules enable to represent correspondences between representations of two adjacent levels, and they shall be automatically generated from the dictionary. Let us just note two issues with the definition of rules for the moment:

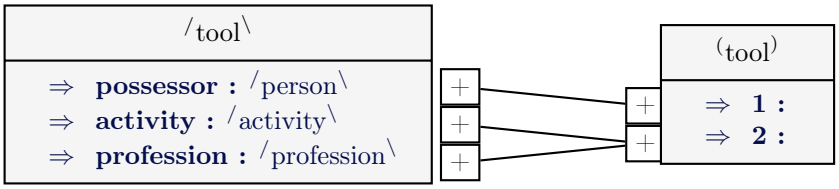
- in the correspondence between actantial structures of /peigne\ and (peigne), the ASlot 2 of (peigne) may correspond either to the ASlot *combedhair* or to *combedperson* of /peigne\. One thus need to define two different correspondence rules, one that assumes slot *combedhair* is filled, and one that assumes slot *combedperson* is filled.
- if a LexUT has an optional SemASlot, then one would need two different correspondence rules between its associated DSemUT and SSemUTs: one that assumes the ASlot is filled and one that assumes the ASlot is not filled.

These remarks are valid not only at the deep-surface semantic interface, and the number of rules would grow in case of several optional ASlots, several split ASlots, or optional split SemASlots for instance. The semantic web SPARQL query language has OPTIONAL and UNION constructors that we could draw inspiration from to extend the definition of rules so as to factorize these cases. Now for the purpose of our presentation we will hold on the simple definition of rules given above, and rely on the fact that we do have means to compute all the possible correspondence rules.

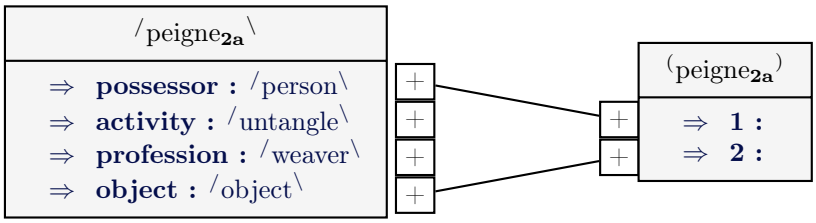
11.2 Rules in the Meaning-Text Theory

Rules enable to represent correspondences between representations of two adjacent levels, and some shall be automatically generated from the government pattern. In this section we will define the correspondence between ASlots of a DSemUT and ASlots of a SSemUT.

Suppose Sophie wants to represent the correspondence between the deep and surface semantic actant slots for TOOL and PEIGNE$_{\mathbf{2A}}$. Sophie opens a new dedicated tab for each of these tasks. The content on the tab is: on the left a box for the DSemUT with its actantial structure, and on the right a box for the SSemUT with its actantial structure. A button is situated in front of each ASlot as illustrated in figures 15a and 15b, and Sophie just needs to drag and drop one of these buttons to the other, so as to link deep semantic ASlots with surface semantic ASlots. Every ASlot of a SSemUT must be linked to at least one ASlot of a DSemUT, several in case of split SemASlots.



(a) Illustration of the deep-surface semantic ASlots correspondence for TOOL.



(b) Illustration of the deep-surface semantic ASlots correspondence for PEIGNE$_{\mathbf{2A}}$.

Figure 15: Illustration of the correspondence between the actantial structure of a Surface Semantic Unit Type, and the actantial structure of its associated Deep Semantic Unit Type.

12 Unit Types Definitions

12.1 Definition of Unit Types Definitions

We formalize the notion of *definition* of a Primitive Unit Type (PUT) and include a set of PUTs definitions in the definition of the unit types hierarchy. Definitions are of special interest to represent lexicographic definitions of a LexUT L, which corresponds to the definition of its associated DSemUT /L\. Informally, a definition defines an equivalence between two λ-UG defined over the same support. One of them has a central free unit node typed with the defined PUT and some of its ASlots filled by free unit nodes. The other λ-UG is called the *expansion* of t. There is no circumstantial triple in these two λ-UG because they must not be part of the lexicographic definition of a LexUT.

Definition 12.1. A definition D_t of a PUT t is a triple $D_t \stackrel{\text{def}}{=} (D_t^-, D_t^+, \kappa)$ where:

- $D_t^- = \{u_t^-, v_1^-, \ldots, v_n^-\}(U^-, \boldsymbol{l}^-, A^-, \varnothing, \varnothing)$ contains only free unit nodes;
- u_t^- is called the the *central unit node* of D_t^-, and is typed $\{t\}$;
- the actantial triples of A^- are of the form (u_t^-, s_i, v_i^-) where the set of s_i is a subset of the ASlots of t;
- for all $(u_t^-, s_i, v_i^-) \in A^-$, the type of v_i^- corresponds to the signature of t for its ASlot s_i;
- $D_t^+ = \{u_t^+, v_1^+, \ldots, v_n^+\}(U^+, \boldsymbol{l}^+, A^+, \varnothing, \varnothing)$ is called the *expansion* of t;
- κ is a bijection from $\{u_t^-, v_1^-, \ldots, v_n^-\}$ to $\{u_t^+, v_1^+, \ldots, v_n^+\}$, such that $\kappa(u_t^-) = u_t^+$, and for all i, $\kappa(v_i^-) = v_i^+$;
- the type of u_t^+ is called the *genus* of t and is denoted $genus(t)$.

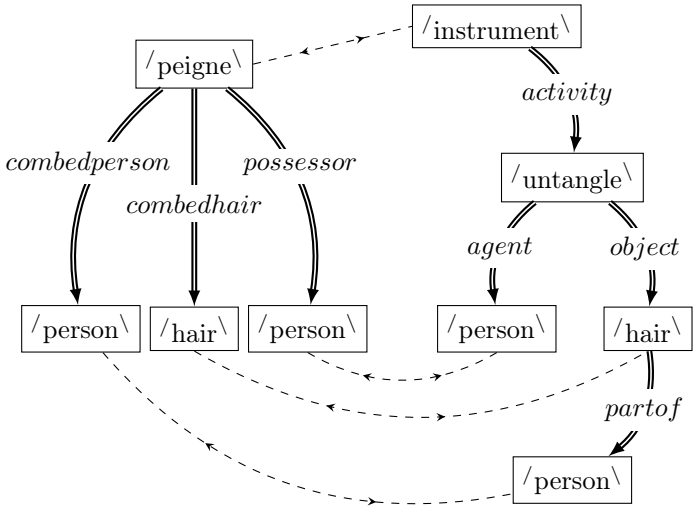


Figure 16: Lexicographic definition of PEIGNE.

Figure 16 is an example of lexicographic definition of PEIGNE: an instrument that a person X uses to untangle the hair Y_1 of a person Y_2.

Intuitively, a definition corresponds to two reciprocal rules: (D_t^-, D_t^+, κ) and $(D_t^+, D_t^-, \kappa^{-1})$. If there is the defined PUT in a UG then one may infer its definition, and vice versa. Again, there is currently an issue with the definitions with optional ASlots. In fact, one would need two different definitions, one with the ASlot filled, and one with the ASlot not filled. For the purpose of our presentation we will hold on the simple definition of definitions given above, and simply represent multiple definitions for a given PUT.

12.2 Unit Types Hierarchy with Definitions of Unit Types

A set of PUTs definitions $\mathcal{D}$ may thus be added to the unit types hierarchy:

Definition 12.2. A hierarchy of unit types, denoted $\mathcal{T}$, is a tuple $\mathcal{T} \stackrel{\text{def}}{=} (T_D, \boldsymbol{S}_{\mathcal{T}}, \boldsymbol{\gamma}, \boldsymbol{\gamma_1}, \boldsymbol{\gamma_0}, C_A, \bot_A^{\sqcap}, \{\boldsymbol{\varsigma}_t\}_{t \in \boldsymbol{T}}, \mathcal{D})$ that enables to construct a pre-ordered set of unit types $\boldsymbol{T}^{\cap}$ with their actantial structure, and with $\mathcal{D}$ being definitions of some PUTs.

Consider a set of definitions $\mathcal{D}$. If a definition explicitly or implicitly refers to the type of unit it defines, then there is a circularity in $\mathcal{D}$.

Definition 12.3 (Circularity)**.** A set of definitions $\mathcal{D}$ has a circularity if and only if there exists a series $t_1, \ldots, t_n$ of PUTs such that $t_1 = t_n$ and for each PUT $t_i, i \in [1..n-1]$, there exists a definition of t_i such that the expansion of t contains a unit node u with $t_{i+1} \in type(u)$.

12.3 Lexicographic Definitions in the Meaning-Text Theory

Lexicographic definitions are to be represented at the deep semantic level, as an equivalence between two deep semantic UGs.

Definition 12.4 (Definition of a unit type, Lexicographic definition of a LexUT)**.** Let $/\mathrm{L}\backslash$ be the DSemUT associated with lexical unit L. The lexicographic definition of L corresponds to the definition of $/\mathrm{L}\backslash$, i.e., a triple $D_{/\mathrm{L}\backslash} \stackrel{\text{def}}{=} (D^-_{/\mathrm{L}\backslash}, D^+_{/\mathrm{L}\backslash}, \kappa)$.

To one definition may thus correspond two reciprocal rules: one that adds $D^{+}_{/\mathrm{L}\backslash}$ to a graph where $D^{-}_{/\mathrm{L}\backslash}$ projects, and one that adds $D^{-}_{/\mathrm{L}\backslash}$ to a graph where $D^{+}_{/\mathrm{L}\backslash}$ projects. If there is the defined PUT in a UG then one may infer its definition, and vice versa.

Let us sketch how Sophie may define the lexicographic definition of PEIGNE$_{\mathbf{2A}}$, i.e, the definition of /peigne$_{\mathbf{2a}}$\.

The starting point is the box that represents the actantial structure of /peigne$_{\mathbf{2a}}$\ as illustrated in figure 6d.

1. Sophie first drags and drops some ASlots outside the box. This enables to make explicit for instance that /untangle\ has two obligatory ASlot. The result of this process is illustrated in figure 17a.

2. Sophie may then drag the ASlot *agent* of /untangle\ and drop it over the box of /person\. This merges participants as illustrated in figure 17b.

3. The *object* of /peigne$_{\mathbf{2a}}$\ and the *fibres* of /untangle\ must be linked by a meronymy relation. For the sake of illustration, we assume there exists a DSemUT /partOf\ that carries this meaning. Sophie clicks on a "add a unit node" button, and seeks for /partOf\ in the hierarchy of DSemUTs. A unit node typed /partOf\ is then added as in figure 17c.

4. Sophie drags the *whole* of /partOf\ and drops it over the *object* of /peigne$_{\mathbf{2a}}$\; and drags the *part* of /partOf\ and drops it over the *fibres* of /untangle\. The result of this process is illustrated on figure 17d.

From this graph one may automatically build the definition $D_{/\mathrm{peigne}_{\mathbf{2a}}\backslash} = (D^{-}_{/\mathrm{peigne}_{\mathbf{2a}}\backslash}, D^{+}_{/\mathrm{peigne}_{\mathbf{2a}}\backslash}, \kappa)$ of /peigne$_{\mathbf{2a}}$\ such as defined in definition 12.4. This definition is illustrated in figure 18.

Part V
Conclusion

We thus studied how to formalize, in the sense of knowledge engineering, the ECD, in order to represent, manipulate, query, and reason linguistic knowledge.

We showed that both semantic web and conceptual graphs formalisms are not adapted to represent knowledge of the ECD while ensuring good computational properties. We hence justified the introduction of the new UGs graph-based knowledge representation formalism.

The linguistic predicates are represented by unit types, and are described in a unit types hierarchy. It consists in a minimal set of mathematical objects that allows to construct a pre-ordered set of unit types with actantial structures. The actantial structure of a unit type is composed of actant slots that may be optional, obligatory, or signed, and that are signed. Moreover, a unit type inherits and possibly specialize the actantial structure of its parents. The strong coherence in the unit types hierarchy justifies the introduction of a deep semantic representation level that is deeper than the semantic level, and in which one may represent the actual meaning of LexUT. The deep semantic unit type $^{/}\mathrm{L}^{\backslash}$ associated with a LexUT L has ASlots that are symbolized by semantic roles, and that correspond to participants of the linguistic situation denoted by L which are SemASlots of L or of LexUTs whose meaning is less specific than L.

Circumstantial relations are another kind of dependency relation that are described in a hierarchy. Along with a set of unit identifiers these two structures form a UGs support on which UGs may be defined. As UG have an underlying oriented labelled graph, one could introduce the notion of UG homomorphism which is useful to define the applicability of rules and the entailment problem.

We introduced the semantics of UGs and we posed the entailment problem for UGs. A UGs along with the support on which it is defined represents explicit knowledge, and additional knowledge may be inferred. We introduced a set of entailment rules that one may use to compute the closure $cl(G)$ of a UG G, i.e., make explicit all of the knowledge that is implicit. We then defined the entailment problem of H by G as a directed labelled graphs homomorphism problem between H, and the closure of G: $cl(G)$. In case $cl(G)$ is finite, the entailment problem is thus NP-complete.

Rules enable to specify correspondences between ASlots of corresponding unit types at adjacent representation levels. We illustrated our approach with a scenario at the deep-surface semantic level interface, and showed how split ASlots shall be dealt with.

The lexicographic definitions of LexUTs shall be represented at the deep semantic level. We introduced the lexicographic definition of LexUTs as definitions of their associated DSemUT. We detailed an application scenario in the context of the RELIEF project: a lexicographer may manipulate nodes so as to little by little construct a deep semantic graph that represents the decomposition of the deep semantic unit type associated with the defined LexUT.

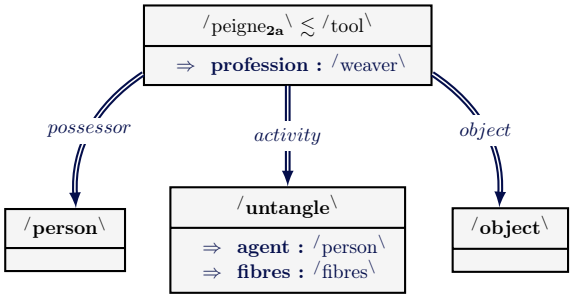
In this research report we also sketched two directions for future research:

- Many rules may be needed to represent correspondences between the deep semantic and the semantic representation levels in case some SemASlots are optional or split. More research is needed to adapt the SPARQL OPTIONAL and UNION constructors in these cases. The same can be said about definitions of DSemUTs that have optional ASlots.

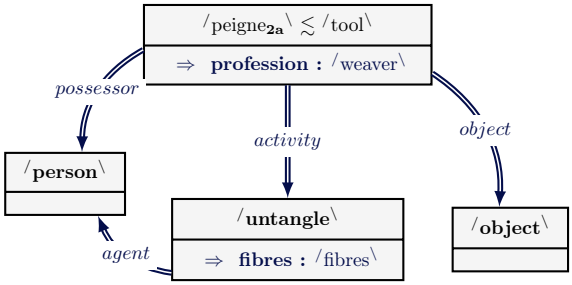
- The closure may be infinite for finite UGs. If that occurs it makes the closure undecidable, along with the entailment problem. We are currently working of the definition of restrictions of the unit types hierarchy and the set of definitions in order to ensure that any UG has a finite closure.

We are also currently working on factorization of rules that would enable us to represent lexical functions links, and on a syntax based on semantic web formalisms standards to enable the standardized exchange of knowledge of the ECD.

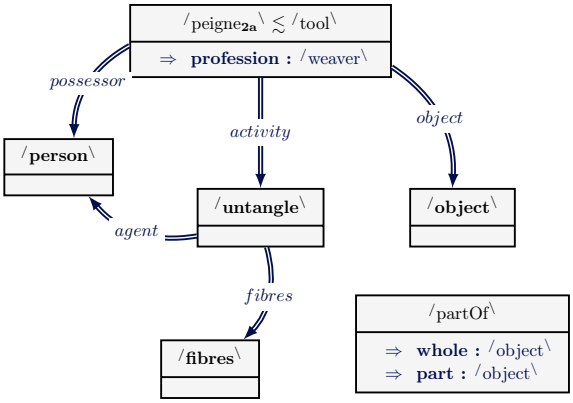
Thanks We would like to warmly thank S. Kahane, A. Polguère, C. Boitet, and anonymous reviewers of the different papers that we wrote about Unit Graphs.



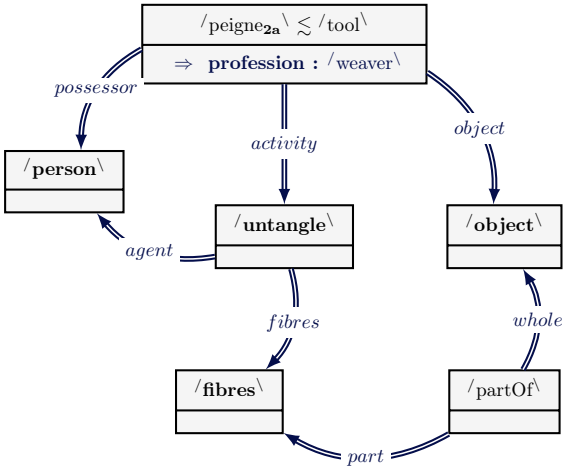
(a) Interesting participants of the definition of /peigne**2a**\ may be given a node by drag and drop.



(b) One may merge participants using drag and drop.



(c) One may add nodes in the definition.



(d) Complete definition of /peigne**2a**\.

Figure 17: Different steps in the definition of the Deep Semantic Unit Type /peigne**2a**\.

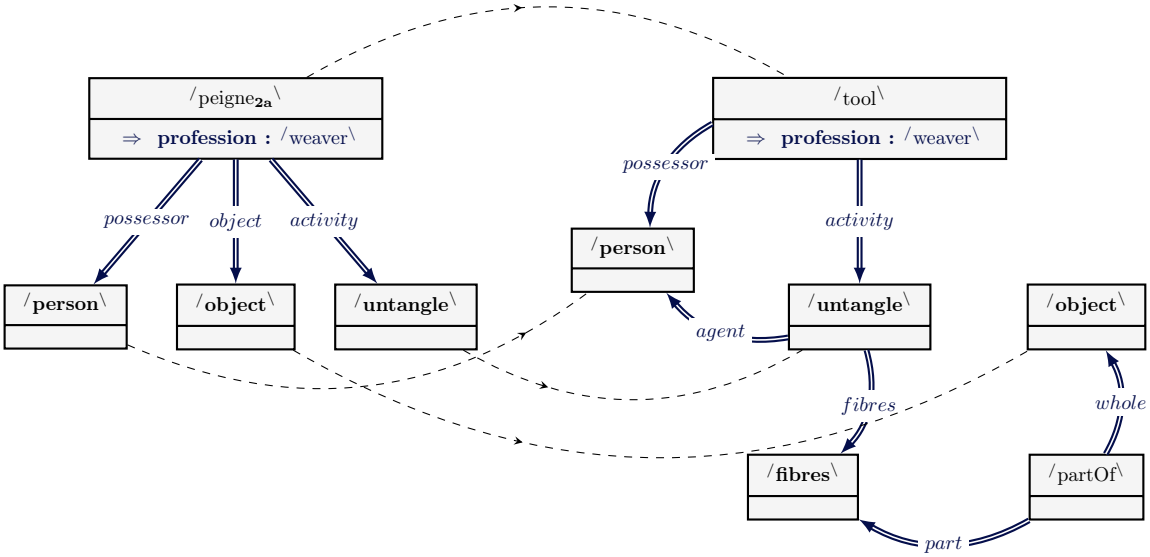


Figure 18: Illustration of the definition $D_{/\text{peigne}_{\mathbf{2a}}\backslash} = (D^-_{/\text{peigne}_{\mathbf{2a}}\backslash}, D^+_{/\text{peigne}_{\mathbf{2a}}\backslash}, \kappa)$ of $/\text{peigne}_{\mathbf{2a}}\backslash$. $D^-_{/\text{peigne}_{\mathbf{2a}}\backslash}$ is on the left, the expansion $D^+_{/\text{peigne}_{\mathbf{2a}}\backslash}$ on the right, and the dashed links represent the mapping κ.

References

Alonso Ramos, M. (2003). Hacia un Diccionario de colocaciones del español y su codificación. *Lexicografía computacional y semántica*, pages 11–34.

Apresian, J., Boguslavsky, I., Iomdin, L., Lazursky, A., Sannikov, V., Sizov, V., and Tsinman, L. (2003). ETAP-3 linguistic processor: A full-fledged NLP implementation of the MTT. In *First International Conference on Meaning–Text Theory (MTT'2003)*, pages 279–288.

Baget, J.-F., Croitoru, M., Leclère, M., Mugnier, M.-L., and Gutierrez, A. (2010). Translations between RDF(S) and conceptual graphs. *ICCS'10: 18th International Conference on Conceptual Structures - From Information to Intelligence*, pages 28–41.

Barque, L., Nasr, A., and Polguère, A. (2010). From the Definitions of the 'Trésor de la Langue Française' To a Semantic Database of the French Language. In Fryske Akademy, editor, *Proceedings of the XIV Euralex International Congress*, Fryske Akademy, pages 245–252, Leeuwarden, Pays-Bas. Anne Dykstra et Tanneke Schoonheim, dir.

Barque, L. and Polguère, A. (2008). Enrichissement formel des définitions du Trésor de la Langue Française informatisé (TLFi) dans une perspective lexicographique. *Lexique*, 22.

Boguslavsky, I. (2011). Semantic Analysis Based on Linguistic and Ontological Resources. In Boguslavsky, I. and Wanner, L., editors, *Proceedings of the 5th International Conference on Meaning-Text Theory (MTT'2011)*, pages 25–36, Barcelona, Spain. INALCO.

Boguslavsky, I., Iomdin, L., and Sizov, V. (2004). Multilinguality in ETAP-3: reuse of lexical resources. In Sérasset, G., editor, *Proc. COLING 2004 Multilingual Linguistic Ressources*, pages 1–8, Geneva, Switzerland. COLING.

Bohnet, B. and Wanner, L. (2010). Open source graph transducer interpreter and grammar development environment. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, pages 19–21, Valletta, Malta. European Language Resources Association (ELRA).

Chein, M. and Mugnier, M.-L. (2008). *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*. Springer-Verlag New York Incorporated.

Corby, O., Dieng, R., and Hébert, C. (2000). A Conceptual Graph Model for {W3C} Resource Description Framework. In Ganter, B. and Mineau, G. W., editors, *Conceptual Structures: Logical, Linguistic, and Computational Issues*, number 1867 in Lecture Notes in Computer Science, pages 468–482. Springer Berlin Heidelberg.

Kahane, S. and Polguère, A. (2001). Formal foundation of lexical functions. In *Proceedings of ACL/EACL 2001 Workshop on Collocation*, pages 8–15.

Krisnadhi, A., Maier, F., and Hitzler, P. (2011). OWL and Rules. *Reasoning Web. Semantic Technologies for the Web of Data*, pages 382–415.

Leclère, M. (1998). Raisonner avec des définitions de types dans le modèle des graphes conceptuels. *Revue d'intelligence artificielle*, 12(2):243–278.

Lefrançois, M. and Gandon, F. (2011a). ILexicOn: Toward an ECD-Compliant Interlingual Lexical Ontology Described with Semantic Web Formalisms. In Boguslavsky, I. and Wanner, L., editors, *Proceedings of the 5th International Conference on Meaning-Text Theory (MTT'2011)*, pages 155–164, Barcelona, Spain. INALCO.

Lefrançois, M. and Gandon, F. (2011b). ULiS: An Expert System on Linguistics to Support Multilingual Management of Interlingual Semantic Web Knowledge bases. In Elena Montiel-Ponsoda John McCrae, P. B. and Cimiano, P., editors, *Proc. of the 9th International Conference on Terminology and Artificial Intelligence (TIA 2011)*, volume 775, pages 108–114, Paris, France. CEUR-WS.

L'Homme, M.-C. (2008). Le DiCoInfo: Méthodologie pour une nouvelle génération de dictionnaires spécialisés. *Traduire*, (217):78–103.

Lux-Pogodalla, V. and Polguère, A. (2011). Construction of a French Lexical Network: Methodological Issues. In *Proceedings of the International Workshop on Lexical Resources*, Ljubljana.

Mel'čuk, I. (1996). Lexical Functions: A Tool for the Description of Lexical Relations in a Lexicon. In Wanner, L., editor, *Lexical Functions in Lexicography and Natural Language Processing*, pages 37–102. Benjamins Academic Publishers, Amsterdam/Philadelphia.

Mel'čuk, I. (2004). Actants in Semantics and Syntax I: Actants in Semantics. *Linguistics*, 42(1):247–291.

Mel'čuk, I. (2006). Explanatory combinatorial dictionary. *Open Problems in Linguistics and Lexicography*, pages 225–355.

Mel'čuk, I., Arbatchewsky-Jumarie, N., Iordanskaja, L., Mantha, S., and Polguère, A. (1999). *Dictionnaire explicatif et combinatoire du français contemporain. Recherches lexico-sémantiques IV*. Les Presses de l'Université de Montréal, Montréal, Canada.

Mugnier, M. and Chein, M. (1996). Représenter des connaissances et raisonner avec des graphes. *Revue d'intelligence artificielle*, 10(1).

Polguère, A. (2000). Une base de données lexicales du français et ses applications possibles en didactique. *Revue de Linguistique et de Didactique des Langues*, 21:75–97.

Polguère, A. (2009). Lexical systems: graph models of natural language lexicons. *Language resources and evaluation*, 43(1):41–55.

Polguère, A. (2011). Classification sémantique des lexies fondée sur le paraphrasage. *Cahiers de lexicologie*, 98:197–211.

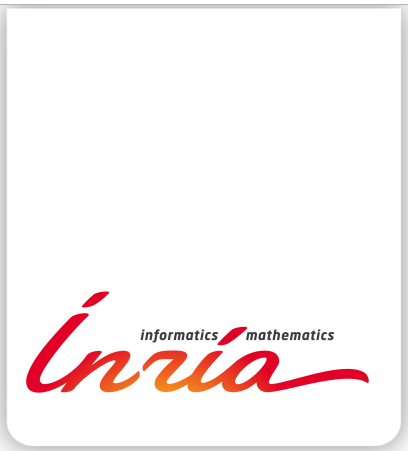
Rudolph, S. (2011). Foundations of description logics. *Reasoning Web. Semantic Technologies for the Web oData*, pages 76—-136.

Sérasset, G. (1997). Le projet NADIA-DEC: vers un dictionnaire explicatif et combinatoire informatisé. *La mémoire des mots, Ve journées scientifiques du réseau LTT*, 97:149–159.

Sowa, J. (1989). Using a lexicon of canonical graphs in a semantic interpreter. In *Relational models of the lexicon*, pages 113–137. Cambridge University Press New York, NY, USA.

Sowa, J. F. (1984). *Conceptual structures: information processing in mind and machine*. System programming series. Addison-Wesley Pub., Reading, MA.

Tesnière, L. (1959). *Éléments de syntaxe structurale*. C. Klincksieck (Colombes, Impr. ITE).



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399